A Knowledge-Based Activity Representation for Shared Autonomy Teleoperation of Robotic Arms

Mohamed Behery
341054

August 10, 2016

First Examiner: Prof. Gerhard Lakemeyer, Ph.D.
Second Examiner: Prof. Dr.-Ing. Alin Olimpiu Albu-Schäffer

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

"Any mystery can be solved through the application of knowledge and unrelenting effort."

– James Luceno, Darth Plagueis, 2012.

Acknowledgements

This thesis was written in the Robotics and Mechatronics Center (RMC) of the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt; DLR). A lot of people helped make this work possible. I would like to thank Prof. Lakemeyer the head of the Knowledge Based Systems Group (KBSG) of the RWTH-Aachen University and Prof. Albu-Schäfer, head of the institute of robotics and mechatronics at the RMC for giving me the opportunity of doing this thesis under their supervision. Special thanks are in order to Dr. Schiffer for all his guidance and support during the period of the thesis. I am also grateful for my mentors, supervisors, and teammates, Mr. Leidner and Mr. Vogel for their guidance, support, and motivation throughout the project as well as for creating a relaxed and enjoyable work environment. My sincere gratitude goes to my colleagues and friends at the DLR who provided a warm and friendly atmosphere, and who made my work such a delightful experience.

Last but not least, I would like to thank my family for their unwavering support and motivation. They always provide a shoulder to lean on in times of need and for that I'm eternally grateful.

Abstract

Physical impairment caused by illness, disability, or age, limits a person's independence. Performing Activities of Daily Living (ADLs), such as eating and drinking, can be challenging for people with physical impairment since they have no muscle control. Recent studies investigated employing teleoperated robotic arm/ hand systems to facilitate the performance of ADLs by physically impaired users. Robot teleoperation using Brain Computer Interface (BCI) empowers users with the ability to perform ADLs. However, it is difficult to map the signals received from the BCI into commands for controlling a 7 Degrees of Freedom robot arm, therefore a user will find it difficult and uncomfortable to perform such activities using a robot arm controlled with a BCI. To overcome this, this thesis introduces the Constraint-based Activity REpresentation (CARE) framework; a framework for shared autonomy teleoperation of a robotic arm using a BCI. The CARE framework defines activities in an object centric approach, where an object declares a set of symbolic and geometric constraints to be applied while performing different manipulation actions that might involve other objects. The framework assists the user to satisfy these constraints throughout the activity and compensates for the low dimensional signals obtained from the BCI. As a demonstration, the CARE framework was employed to assist an operator in different ADLs, such as pick and place, pouring, door opening, and wiping. The demonstrations were conducted using a simulation of the DLR EDAN assistance robot controlled using a SpaceMouse.

Contents

List of Figures

Abbreviations

ADL Activities of Daily Living. 1–3, 9–12, 17, 19, 51–53, 59, 61, 62

BCI Brain Computer Interface. 2, 3, 9, 10, 51, 61, 62

BFGS Broyden Fletcher Goldfarb Shanno algorithm. 12

CARE Constraint-based Activity REpresentation. 2, 3, 19, 21, 51, 53, 55, 61

DH Denavit-Hartenberg. 6

DLR German Aerospace Center (Deutsches Zentrum fur Lüft- und Raumfahrt). 2, 19, 51

DoF Degrees of Freedom. 5, 6, 9, 62

EM Expectation Maximization. 17

EMG Electromyography. 2, 9, 19

GUI Graphical User Interface. 1, 10, 13

HMD Head Mounted Display. 10

IK Inverse Kinematics. 13, 33

LWR Light-Weight Robot. 2, 19, 51, 52

MEMM Maximum Entropy Markov Model. 16

MTM Manipulation Task Model. 12

ODB Object DataBase. 20, 27, 36, 47, 49, 50

PDDL Planning Domain Definition Language. 13, 27

RB Roaming Beats. 16

ROI Region Of Interest. 27, 45, 47

SMA Spinal Muscular Atrophy. 1, 61

SUS System Usability Scale. 53, 55

UAV Unmanned Aerial Vehicle. 11

1 Introduction

Activities of Daily Living (ADL) form a major challenge for people with tetraplegia. Tetraplegia can be caused by spinal cord injuries or a disease affecting the brain or the spinal cord. The lack of motor control in the limbs makes activities, such as pouring a cup of water, impossible for a person with tetraplegia without external help. For example, Spinal Muscular Atrophy (SMA) is a genetic disease that attacks the upper spinal cord cells responsible for communicating with the muscles. People with SMA lose control of the muscles in the arms and legs, and sometimes the muscles responsible for breathing and swallowing. They lose their physical strength and mobility due to the atrophy of the muscles. As a result, these people are bed-bound and rely on wheelchairs and a 24-hour care and assistance from another person [41].

An autonomous assistance robot could perform ADLs of varying complexity where commands are communicated to the robot through input devices and interfaces such as a keyboard or a Graphical User Interface (GUI) [2]. But people with tetraplegia cannot use such input devices and interfaces due to their lack of motor control, not to mention that a fully autonomous robot deprives its users of flexibility and full control over the activities. This eliminates fully autonomous

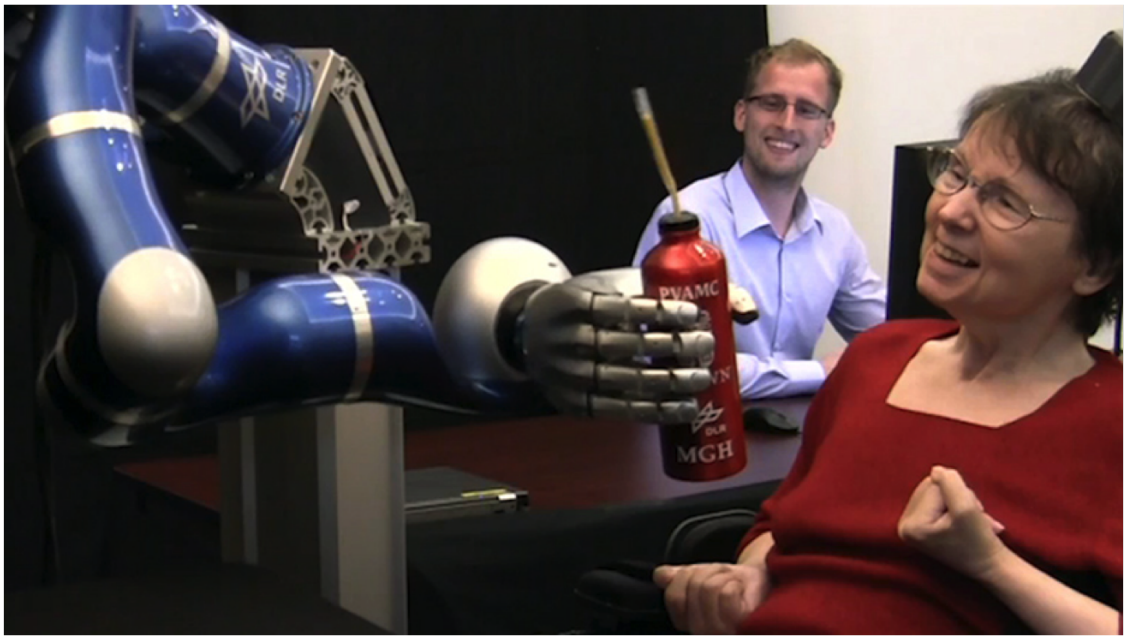


Figure 1: People with tetraplegia have little to no muscular control, they lose motor control over their extremities and therefore rely on the assistance of a caregiver their everyday needs. The figure shows a person with tetraplegia controlling a DLR LWR arm using EMG to pick a bottle,

robotic assistance from viable alternatives for the human assistance to people with tetraplegia.

In contrast, a teleoperated robotic arm gives its operator control and flexibility while performing tasks. But teleoperation requires an input method that communicates commands to the arm. Wolpaw et al. [44] show that brain activity can be monitored and used for communication with the external world using a Brain Computer Interface (BCI). Hochberg et al. [15] demonstrated that people with disabilities, unable to use traditional input devices, such as computers and joysticks, are able to use a BCI to control a robotic arm and perform three dimensional manipulations, which allows these people to perform ADLs.

The German Aerospace Center (Deutsches Zentrum fur Lüft- und Raumfahrt) (DLR) introduced the EDAN assistance robot; a DLR/HIT hand mound on a DLR Light-Weight Robot (LWR) arm mounted on a wheel chair. People with tetraplegia can teleoperate EDAN through a BCI or Electromyography (EMG) signals to perform ADLs as seen in Figure 1. However, using a BCI-controlled robotic arm, ADLs (such as grasping an object) can be hard, time-consuming, and inaccurate. This is due to the difficulty in decoding brain signals as explained by Green and Kalaska [12].

Muelling et al. [29] introduced a framework to offer assistance to people with tetraplegia through shared autonomy of a teleoperated robot arm. The framework addresses the challenges of controlling a robot arm using a BCI. While performing a task, the framework in [29] infers the user's intention and moves the robot's end effector in a pose that facilitates the task. For example, when approaching a bottle, the robot arm adjusts its pose to a feasible grasping position. Instead of approaching the bottle directly, the framework reasons about the type of object (in this case, a bottle) and the direction from which the user approaches it to find the intended grasp pose.

To use this setup for performing ADLs, generic action descriptions are needed. These action descriptions are difficult to find if the actions involve complex object handling mechanisms or the use of tools, which is the case for most ADLs. The main goal of this thesis is to find a representation for manipulation actions that allows shared autonomy teleoperation.

Thesis Contributions

In order to make life easier for people with tetraplegia and reduce their dependence on caregivers, this thesis aims to facilitate the teleoperation of a robot arm by people with tetraplegia to perform ADLs without external help. This thesis introduces the Constraint-based Activity REpresentation (CARE) framework which uses shared autonomy teleoperation to enhance the commands of a person with tetraplegia performing ADLs without depriving them of control over the activity. This is achieved by the contributions of the thesis:

1. A constraint based representation of manipulation actions for shared autonomy teleoperation, where an object defines constraints for interaction with other objects.
2. A shared autonomy system for enhancing an operator's commands to satisfy the constraints provided by each object while performing an activity.

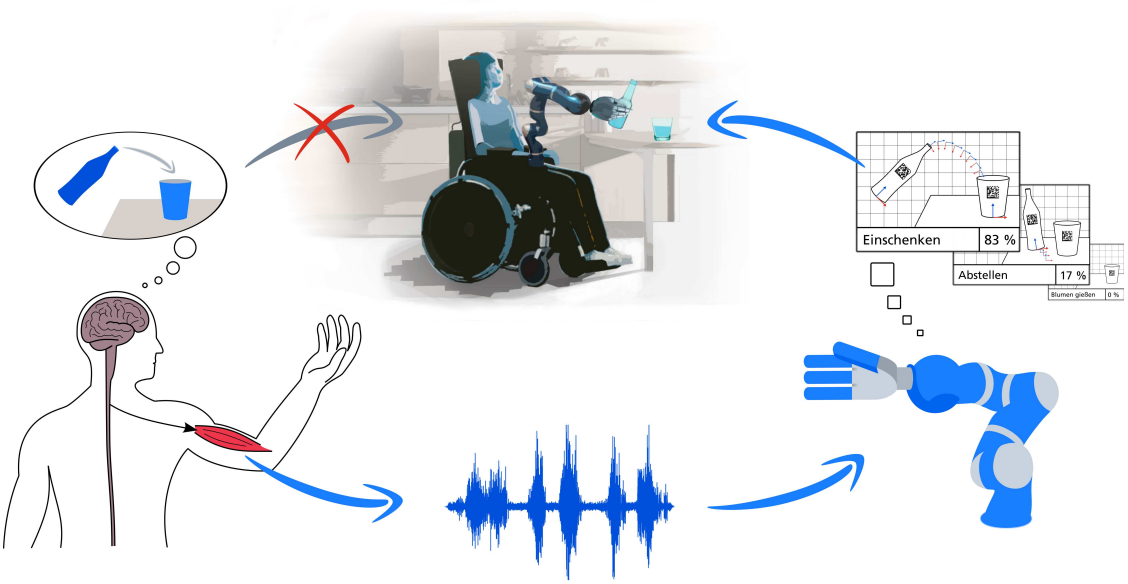


Figure 2: The work flow of the framework. Since it is not possible to find the user's intention directly from the Brain Computer Interface (BCI) and perform it, the system uses the geometric and symbolic world states as well as how the user wants to move the arm, and moves the robot arm based on that information. The motion resulting from the reasoning facilitates the user's intended activity.

3. An algorithm for user intention inference.

Figure 2 shows the concept of the framework. The performance of ADLs using BCI is not an easy task since BCIs cannot infer the user's intended activity. In this work, the user's activity is inferred based on the world state and user commands. End effector trajectories are calculated based on the inferred activity making an ADL a simpler task to perform. A trajectory defines how the end effector moves and rotates as a response to user commands based on the inferred activity, this means the user doesn't have to worry about all the dimensions of motion while teleoperating the arm, making ADL performance effortless.

The effectiveness of the CARE framework is demonstrated by using it to perform different ADLs. Users controlled a simulation of the arm to perform different tasks involved in ADLs. The tasks included pick, place, pouring, wiping, and drawer opening.

Thesis Structure

This thesis reports how the above contributions were achieved and evaluated. The remainder of this thesis is structured as follows, Section 2 gives a brief introduction to the relevant terminology used in within this thesis. Section 3 mentions the work related to this thesis as well as work that inspired the ideas presented in this thesis. Section 4 gives an overview of the approach and demonstrates how it works in different scenarios. Section 5 shows the testing and evaluation of the framework and the results of the thesis. Finally, Section 6 gives a brief summary of the work done during the thesis and discusses the possible future work to be performed in continuation to this work.

2 Background and Foundations

This section presents the terminology relevant to this thesis commonly used in the field of robotics. This terminology is used throughout the report to describe the practical aspects of the work presented here.

2.1 Basic Terminology

In order to teleoperate a robotic arm, one needs a way to communicate how the arm should move. Depending on the control algorithm, this could be a new manipulator pose (location and orientation) or a manipulator pose with a force to be applied while doing this motion. In this work a transformation matrix is used to indicate the manipulator pose during motion.

A transformation matrix has the form shown in (1), the upper-left sub-matrix contains rotations around the x, y, and z-axes, the upper-right one contains translations in the x, y, and z directions; the 6 Degrees of Freedom (DoF) of the Cartesian space. The lower left and right sub-matrices contain perspective changes from one coordinate system to another and a scaling factor, respectively. They are set to [0 0 0] and 1 respectively when used in robot manipulation [4] as seen in (2).

$$A = \left[\begin{array}{c:c} \text{Rotation Matrix} & \text{Translation Matrix} \\ \hdashline \text{Perspective Transformation} & \text{Scaling Factor} \end{array}\right] \tag{1}$$

Figure 3: The DLR light-weight robot arm, A redundant robot arm with 7 Degrees of Freedom (DoF). Fitted with the DLR/HIT hand.

$$A = \left[\begin{array}{ccc:c} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{2}$$

The frame (a coordinate set) of an object (or an end-effector) is a 4×4 homogeneous transformation matrix, as shown in (2), showing the location and orientation of the object, relative to the base reference coordinates of the robot. In high level robot programming, the frame of an object is used for the localization, i.e. it defines the pose of the object, while its *toolframe* is used to localize the point of interaction. For example, the frame of a door may be centered on one of the corners of the door while the toolframe would be centered on the door knob.

A redundant robotic arm (such as the DLR Light-weight Robot arm used for the simulation and testing of this work) is a robotic arm whose joint space has a higher dimensionality than the task space. Figure 3 shows the DLR Light-weight Robot (LBRIII) arm; a redundant robot arm with seven joints (7 DoF).

2.2 Forward and Inverse Kinematics

A manipulator pose is achieved by a robot arm through changing the configuration of the arm joints. Before deciding to move the manipulator to a new pose, it is necessary to check whether or not the manipulator can reach this pose before sending it to the control algorithm. This section shows how a robot arm is modeled in order to check whether joint angles exist that lead to a specific manipulator frame.

A robot arm is modeled as a chain of links connected by joints from the base to the end-effector (or hand). The position and orientation of a joint relative to the base can be found by applying a set of transformations from the base to the joint in question. In such a chain, the Denavit-Hartenberg (DH) convention describes the relative transformation between a frame and its predecessor [4, 7]. The DH convention assumes that each joint i has a coordinate set, such that:

- z_{i-1} is parallel to the axis of motion of joint i.
- x_i is normal to z_{i-1} and is pointing away from it.

The transformation between the frames of joint i and joint $i-1$ is defined using four parameters; θ_i, s_i, a_i, and α_i as follows:

- θ_i : The angle between x_{i-1} to x_i measured around the z_{i-1} axis in the clockwise direction.
- s_i : The distance, along the z_{i-1} axis, between the frame of joint $i-1$ and the intersection of x_i and z_{i-1}.

- a_i : The shortest distance between z_{i-1} and z_i axes.
- α_i : The offset angle between z_{i-1} and z_i.

The transformation matrix from the ith coordinates to the $i-1$th coordinates can be described using these four parameters, the general form of this transformation matrix is provided in [31] as (3):

$$A_{(i-1)i} = \left[\begin{array}{ccc:c} cos\ \theta_i & -cos\ \alpha_i\ sin\ \theta_i & sin\ \alpha_i\ sin\ \theta_i & a_i\ cos\ \theta_i \\ sin\ \theta_i & cos\ \alpha_i\ cos\ \theta_i & -sin\ \alpha_i\ cos\ \theta_i & a_i\ sin\ \theta_i \\ 0 & sin\ \alpha_i & cos\ \alpha_i & s_i \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{3}$$

A total transformation matrix T is used to transform between hand and base coordinates for a robot arm with n joints, defined in (4)

$$T = A_{01}\ A_{12}\ A_{23}\ \ldots\ A_{(n-1)n} \tag{4}$$

Since the parameters s_i, a_i, and α_i are determined by the geometry of the arm, the configuration of a robot arm can be represented as the set of joint angles $\mathbf{q} = [\theta_0,\ \theta_1,\ \ldots,\ \theta_n]$. Equation (4) can be transformed into a function of $\mathbf{q}$ as seen in

$$T(\mathbf{q}) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_{01}(q_1)\ A_{12}(q_2)\ A_{23}(q_3)\ \ldots\ A_{(n-1)n}(q_n). \tag{5}$$

The forward (direct) kinematics problem is the problem of finding the frame of the end-effector of the robot given some joint configuration. The problem is straight forward and can be represented using the form of (6) [9], such that $\mathbf{x}$ is a 4×4 homogeneous transformation matrix showing the translation and rotation in Cartesian space, and $\mathbf{q}$ is a n-dimensional vector representing the joint angles, where n is the number of joints. f is a mapping function from the joint space (R^n) to task/Cartesian space (R^3).

$$\mathbf{x} = f(\mathbf{q}) \tag{6}$$

A more practical problem is inverse kinematics; the problem of finding the joint configuration corresponding to a given end-effector frame. This is useful in many robotic tasks and applications such as pick and place, cleaning, welding, ... etc. This can be modeled using (7).

$$\mathbf{q} = f^{-1}(\mathbf{x}) \tag{7}$$

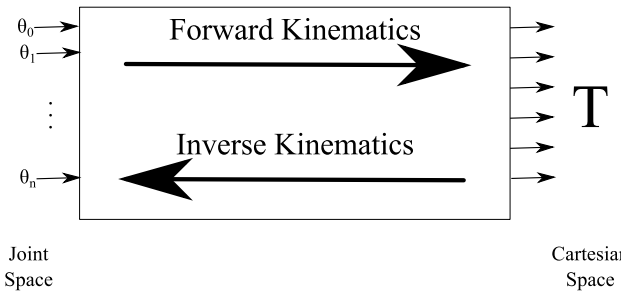


Figure 4: The relation between the forward and inverse kinematics problems. Image reproduced from [20].

This problem can be solved by first finding a solution for q_1 in terms of the elements of T, the middle part of (5), then solving similarly for the rest of the joint angles. This can be done by multiplying both sides of the equation by $[A_{01}(q_1)]^{-1}$. Since $[A_{01}(q_1)]^{-1}[A_{01}(q_1)] = \mathbf{I}$, to obtain (8):

$$[A_{01}(q_1)]^{-1} T = A_{12}(q_2)\, A_{23}(q_3)\, \ldots\, A_{(n-1)n}(q_n) \tag{8}$$

The only unknown on the Left Hand Side (LHS) of (8) is q_1, and the elements of the Right Hand Side (RHS) matrix are either 0, constants, or functions of q_2 through q_n, by equating the elements of the LHS with those of the RHS, q_1 can be found in terms of $r_{11}, \ldots, r_{33}$ and $p_1, \ldots, p_3$. After that $q_2, \ldots, q_n$ can be similarly found [20].

The forward and inverse kinematics problems can be thought of as mappings from joint space to Cartesian space and from Cartesian space to joint space respectively, as seen in Figure 4.

The forward kinematics problem is straight forward and will always have a solution. The inverse kinematics problem, on the other hand, can have more than one solution (possibly an infinite number of solutions), for a redundant manipulator, and can have no solution at all, i.e. there's no joint configuration that will result in the given end-effector pose.

3 State of the Art

This section provides a brief description of the previous work in the field of object representation, action and task description, robotic teleoperation and shared autonomy, as well as intention inference.

3.1 From Direct Control to Shared Autonomy

Performing Activities of Daily Living (ADLs) through teleoperated robots without the assistance of a caregiver is achievable through more than one solution, this is because of the different levels of robotic teleoperation. Teleoperation, as defined by Sheridan [38], commonly refers to direct, continuous control of a robot. Methods for managing robot autonomy in teleoperation (reviewed by Goodrich et al. [11]) include -but are not limited to- Direct Control, Supervisory Control, and Shared Control. The difference between them is the amount of autonomy and human control involved and how they are handled in combination. The following subsections discuss different levels of autonomy involved in robot teleoperation.

3.1.1 Direct Control

Direct user control systems are, as the name implies, systems that apply the user's commands directly to the robot. Brain Computer Interfaces (BCI) and EMG sensors are low dimensional input devices, their signals can't be easily mapped to a high degree of freedom robot arm, therefore using direct control for performing ADLs is not intuitive or easy to use from the users' point of view.

The JACO robotic arm [25] is one of the commercially available 6 Degrees of Freedom (DoF) robot arms offering robotic assistance to people with upper extremity disabilities. It is controlled using a joystick that includes two push buttons for switching between three control modes as follows:

- Moving the manipulator in 3D space without changing the orientation.
- Modify the orientation of the hand while keeping it centered at the same point in 3D space.
- Controlling the grasp.

To control the JACO arm one needs to be able to control the joystick and push its buttons, which is not possible for people with tetraplegia. A directly controlled robot arm needs a high dimensional input signal to correctly perform ADLs, this is not feasible for people who can only communicate with the arm through BCI or EMG signals.

Direct control is not a reliable option in case of lack of motor skills in the upper extremities because BCI signals are difficult to extract and decode [12, 44]. In some cases, compromises

might have to be made between the accuracy and speed when BCI is used for communication with the outside world. This is when the errors in the application are dangerous, such as the task of moving a wheel chair [44]. In other cases, the lack of muscle control might affect signal features used by BCI. Therefore a reliable performance of ADLs using teleoperation of a robot arm requires the inclusion of autonomy in the process.

3.1.2 Supervisory Control

The supervisory control approach is a teleoperation scheme where commands are received from an operator and actions are performed autonomously. It is more reliable in case of lack of motor skills than direct control, since it doesn't rely on the quality of the input signals. But it deprives the operator of the control over the actions as well as the flexibility provided by direct control.

In supervisory control [2, 13, 17, 32, 37], the robot is intermittently programmed by an operator (or multiple operators) while continuously sending information to its operator(s) and autonomously performing the task(s) commanded by the operator(s).

Some teleoperation approaches of this paradigm require the user to use traditional input devices such as a GUI or a touch screen interface [2] and are not suitable for use by a person with tetraplegia. For example, James et al. [17] allows the user to place graphic constructs (affordance templates [13] mentioned in Section 3.2) on 3D sensory data coming from the robot and visualized on a screen to specify goals and their parameters. It offers the ability to complete goals within limited communication between the robot and its operator.

Schroer et al. [37] introduce another framework that assists a disabled person in drinking, the framework uses BCI for user input to control the robotic arm. After receiving at least one start command from the user, the system starts the autonomous execution of the action. They also offer more control to the user with the ability to divide a task into a sequence of actions where the user needs to explicitly give a go-signal after the execution of an action to start the next one. This approach focuses on liquid intake tasks, and does not discuss more complex ADLs.

Petit et al. [32] introduced a framework that uses BCI and a Head Mounted Display (HMD) that allows the embodiment of an operator into a humanoid robot, allowing the operator to control the robot for navigation and interaction with objects. When an object is detected by the vision module, the user can command the robot to interact with that object, the robot then drives autonomously and interacts with the object. Actions are divided into two types, grasping and handing over objects to the user and the interaction with objects is done autonomously with no user control.

Supervisory control approaches offer reliable task completion but they either require an input mechanism unfeasible to people with upper extremity disabilities or deprive their user of flexibility and control while performing a task. This decreases the user satisfaction since users feel more independent when the task execution is completely under their control [44].

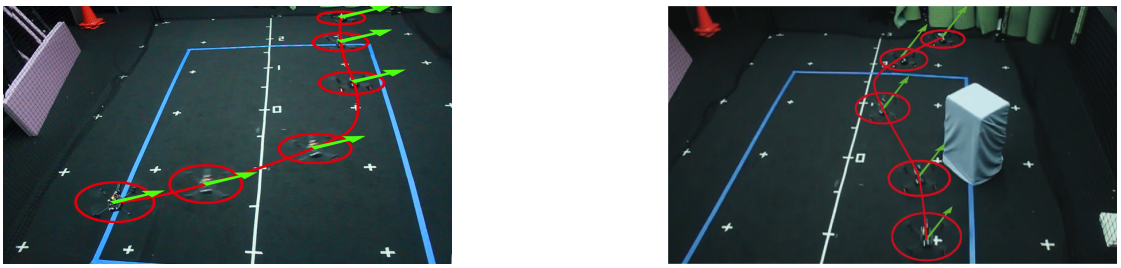


Figure 5: The modified flight paths taken by the UAV. The green arrows indicate the user commands, while the red path indicates the positions of the UAV at different time points after modifying user commands to avoid imminent collisions. Images from [16].

3.1.3 Shared Control

Another teleoperation paradigm is the shared control, it seamlessly combines direct control with autonomy keeping task execution under complete control of the operator while offering less dependence on the input signal quality. They also don't necessarily require the operator to use traditional input devices such as a keyboard or a mouse.

Shared control [16, 26, 29] teleoperation systems are the systems that can change the operator's continuous input to meet a goal or to facilitate the performance of a certain task or action. They are used in a variety of robot control scenarios and are not always dependent on the input device used.

Virtual fixtures have been discussed for use with surgery assistance robots [26], they use open loop virtual fixtures to assist the user in maintaining a direction of motion, and closed loop virtual fixtures to guide the user to a specific location. This is not directly applicable for ADLs since performing ADLs needs more flexibility. However, virtual fixtures are used in this thesis in a way that allows the operator to take full control of the robot arm, offering minimal autonomous assistance when needed such as tilting a bottle when it's close enough to a cup.

Israelsen et al. [16] introduced an algorithm that extrapolates the flight path of an Unmanned Aerial Vehicle (UAV) and applies minimal changes to the operator's commands in order to avoid imminent collisions with obstacles, the modified flight paths are shown in Figure 5. To use this approach in obstacle avoidance for performing ADLs using a robot arm, one needs to account for the different joints and links of the robot arm, an aspect which is not covered in [16].

The framework by Muelling et al. [29] uses a combination of computer vision, user intent inference, and a combination of human and automated control to assist the user in performing ADLs with a teleoperated robot. The authors assume that the user is trying to minimize a cost function while approaching a goal (a grasp pose), commands from the user increase the cost for some poses and decrease it for others. Using this, the system infers the user's intended grasp pose when the environment contains one object. The authors also proposed some extensions,

through which the framework can autonomously assist in performing complex tasks and handle an environment with multiple objects.

The system in [29] uses a model library that has object specific grasp poses and their approach vectors. Each grasp pose is defined by the object type and the direction from which the operator approaches it. When the environment contains multiple objects, the maximum entropy principle [45] is used to decide which object is most likely the intended goal. This allows reasoning about the probability distribution over targets without making commitment beyond the information observed so far. However, they don't consider the symbolic world state or the use of formal action descriptions. They don't consider the difference between a filled or an unfilled cup when considering the pouring activity.

3.2 Object and Action Description Approaches

One of the challenges in the field of knowledge representation and reasoning is describing the tasks or activities a robot is able to perform. Several approaches exist for describing actions based on their geometric properties, symbolic properties, or both. This section gives a brief description of some of these approaches and why they are relevant to this thesis.

Constraint based task description [1] is one of the approaches used in autonomous systems, it defines motion using symbolic constraints represented using geometric relations between the objects, such as the distance between two objects and the approach angle between them.

Lana et al. [21] proposed the Manipulation Task Model (MTM), which is an algebraic approach where a manipulation action is described using a 3-tuple of dual quaternions containing the start pose, dual velocity, and dual force. Where dual velocity quaternions represent the linear and angular velocity, and the dual force quaternion represents force and moment. The model defines the temporal and logical relations between actions. They demonstrate the comprehensiveness of their approach using a window cleaning example where a robot arm uses a wiper to clean two windows placed on two perpendicular walls. This approach is not suitable for shared autonomy scenarios since it would restrict the operator to have a specific starting pose or velocity for each action/ activity.

Jetchev and Toussaint [18] used machine learning approaches to predict trajectories for robots in different situations. The approach uses trajectories that the robot had used in different situations for training and predicts a suitable trajectory when it faces a new situation. A high dimensional feature vector is generated using relative distances and rotations of all objects in the environment, robot joint configuration and the target location. This feature vector is then refined using Broyden Fletcher Goldfarb Shanno algorithm (BFGS) to obtain a lower dimensional vector. This is not applicable in the teleoperation case since the training phase is infeasible due to the lack of data for training. Creating such data would be time consuming since there is no easy teleoperating method for performing ADLs using the robot arm/hand system. Using a planner for autonomously generating such training data consumes less time than teleoperation but generates trajectories that are unintuitive for the operator since they are sometimes inconsistent with a person's natural motion while performing the tasks.

Object centric approaches [5, 13, 23] add action descriptions to the object representations. Dalibard et al. [5] used documented objects, an approach where objects describe how they should be manipulated by autonomous robots. This information is used by a geometric planner along with a simplified model of the robot to generate a collision free plan which is later passed to the Inverse Kinematics (IK) solver controlling the robot. This approach is more suited to humanoid robots than robotic arms, it focuses on avoiding collisions during an activity.

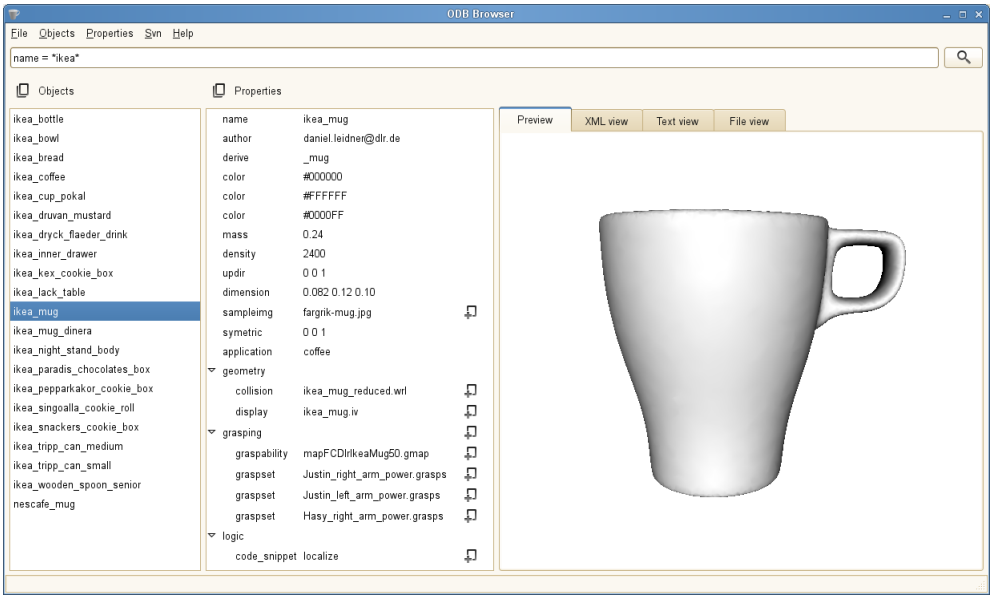
Hart et al. [13] introduced the affordance-templates framework where a human supervisor can place an affordance template on an object in the world using a GUI indicating that this object can afford the action described by that template. This requires a human supervisor with access to a GUI, which is not a feasible input method for people with tetraplegia who are unable to use such input devices, not to mention that the manipulation task will be executed in a fully autonomous manner after the operator specifies the affordance of objects in the scene.

The framework proposed by Leidner et al. [23] uses action descriptions for geometric and symbolic planning for autonomous robots. This is not full applicable to the case of robot teleoperation, however, the work done in this thesis uses the object handling system of [23]. It is divided into two parts; a data storage and a world representation. As seen in Figure 6, the data storage categorizes objects by classes. Each object class contains generic properties as well as a set of action templates for the objects. Properties can contain data related to object classes such as object dimensions, 3D models, and colors, as well as robot dependent information such as grasp sets. The world representation stores the belief state of the robot, the objects described in the data storage are instantiated with concrete parameters from the world.

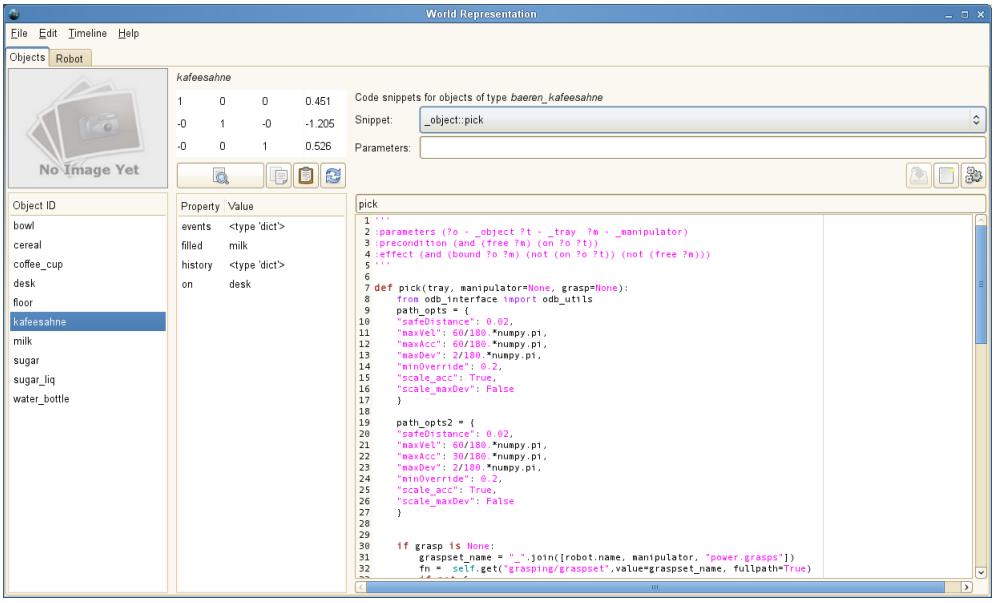
The action templates described in object classes are formed of two segments; a body and a symbolic header. The body of an action contains an abstract action model, it resolves geometric information, loads the object-specific grasp-sets and determines the most appropriate grasp (if needed by the action), and returns a list of operations (robot skills) that will be executed by the robot at run time. This operations can be a combination of moving the manipulator to a specific frame or applying a configuration to an end-effector. The header of an action is written as a complete action description in the Planning Domain Definition Language (PDDL) [10], it defines the symbolic arguments, preconditions and effects of an action. PDDL is used in automated planning to define the planning problem and domain. An example of action headers is shown in Snippet 1.

Action templates are used during the planning phase by a symbolic planner to schedule a set of actions w.r.t. a given goal state. Arbitrary modules are used to simulate the geometric execution of a plan accordingly, if a simulation step fails, then alternative arguments are tested such as trying other grasp poses or selecting a different location to place an object. If all alternatives fail, then the planner backtracks to test another action. This is repeated till the simulation succeeds and then the robot executes the plan.

This approach is not directly applicable in the case of non-delayed (online) teleoperation since the process of planning and simulation is time consuming. Not to mention that having an autonomous execution would deny the user the freedom to control the arm. The work of this thesis, however, relies on the object handling system introduced in [23].



(a)



(b)

Figure 6: The object handling system introduced in [23]. Figure 6a shows the object database which shows different information about an object type, while 6b shows the world representation which contains information about real world objects such as locations, predicate values, tool frame, and has the capability of showing code snippets of actions.

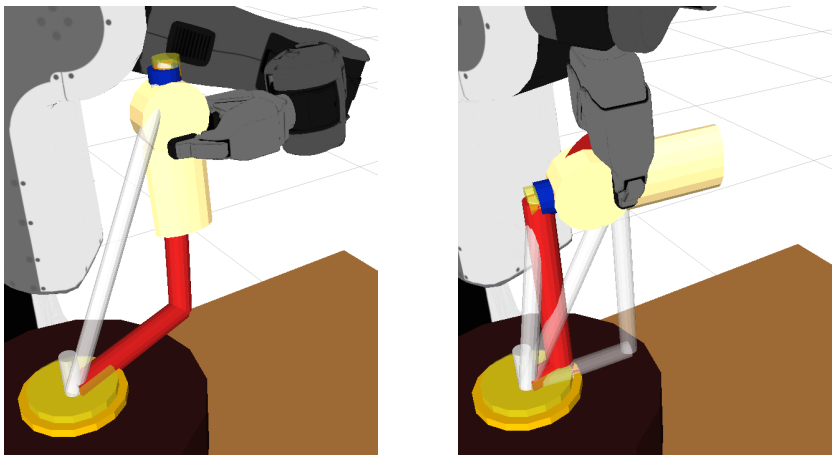


Figure 7: A pouring action modeled using the constraint based action description language, the robot pours the contents of a bottle onto an oven. The modeled task uses four constraints that use four different feature functions. This was modeled using only three geometric features; bottle axis, bottle cap, and the center of the oven. Images from [1].


```
:parameters (?o - _object ?s - _tray ?m - _manipulator)
:precondition (bound ?o ?m)
:effect (and (on ?o ?s) (not (bound ?o ?m)) (free ?m) )
```

Snippet 1: The symbolic headers involved in the action of placing a bottle on a tray-like object. The parameters are descendants of classes object, tray, and manipulator. The only precondition is that the object is bound to the manipulator. As a result of the action the object is no longer bound to the manipulator, the manipulator is free, and the object is on the tray.

Bartels et al. [1] defined a set of geometric features to describe points, line segments, and planes, a set of feature functions describing the relation between these features, such as `perpendicular`, `distance`, and `pointing-at`. A set of these functions is used to define the constraints of an action. Based on this, the robot generates a plan that satisfies these constraints.

Describing an action using a set of constraints allows the programmer to list the success criterion of an action using a set of geometric features for the objects in the scene and a number of relations between these features. The robot tries to perform the movement such that the end configuration satisfies the constraints. Bartels et al. [1] demonstrate the effectiveness of their representation by modeling the different actions involved in making a pancake. Figure 7 shows a robot arm that uses this modeling approach to pour the contents of a bottle onto a pancake oven, Snippet 2 shows a subset of the constraints used in flipping the pancake.

```
constraint{
    tool_feature = spatula-main-axis
    object_features = pancake-center-point
    function = pointing-at
    range [ ... ]
}

constraint{
    tool_feature = spatula-front-axis
    object_features = pancake-rim-line
    function = distance
    range [ ... ]
}
                                        ⋮
```

Snippet 2: A sample of the constraint based movement description language showing a subset of the constraints used to model the action of flipping a pancake. Code snippet from [1].

3.3 Intention Inference

In a direct control scheme, the user's intended activity is not needed by the robot arm, this is because the commands are directly applied to move the end effector. In contrast, in a shared autonomy control paradigm, the robot needs to know what activity the user is performing. Because, the type of assistance offered by the robot depends on the intended activity of the operator. For example, the modification to apply to the user's command in a pick and place task is not the same as the modification applied in a pouring task. In an environment with more than one feasible action, the robot needs to infer the user's activity to be able to offer assistance.

Several approaches exist to infer a person's activity, some algorithms rely on the probability of a certain action following another. Other approaches are based on the state of world; the symbolic and geometric properties of the objects in the environment. Other approaches exist that combine these information to infer a person's intention.

Sung et al. [39] use machine learning approaches to estimate a person's activities. They rely on using an RGB-D sensor to detect the human pose, they use feature vectors based on this data in a hierarchical Maximum Entropy Markov Model (MEMM) to infer the activity a person is doing. After that, a robot offers assistance based on the inferred activity such as clearing a table after a person had finished eating. This method is inapplicable for people with tetraplegia since they are usually bound to a wheel chair or a bed with no muscle control and are therefore unable to change body poses in order to calculate the feature vectors.

Martínez-Pérez et al. [28] introduced using Roaming Beats (RB) [27] to represent objects in the environment. Roaming beats are defined as the time-stamped change in an object's state from motionless to mobile, and changing the location from the base location to a mobile location. They represent an activity using the number of objects involved and the RB of each object, the

time span of an activity, and the maximum waiting time for a beat, they later compare this representation to what is happening in the world to infer a person's activity.

A probabilistic approach for plan recognition was introduced by Tian et al. [40]. They represent actions as strings which means that a plan could be treated as a sentence. The approach relies on learning vector representations for the words/ actions and finding a probabilistic distribution of actions using previously generated plans, then using the *Expectation Maximization (EM)* method to find a set of possibilities/ suggestions for the unobserved actions in a plan. The aim was to provide a plan completion tool suitable for auto-completing search engine queries. The system in [40] was evaluated by training and testing the system in separate worlds individually, i.e. generating plans for each world and using them to train the system for that world. The goal of this thesis is to enable an operator to eventually use the system in their daily life. It is unfeasible to provide similar training data due to the large number of potential activities/ plans a user can perform. In an object centric action definition approach new objects may define new actions, in which case the system will need to be re-trained.

The approaches mentioned so far don't use the symbolic action models based on the preconditions and effects of the actions involved in the plan or symbolic information of the objects in the world. Another approach for intention inference is plan recognition as planning introduced by Ramırez and Geffner [34]. They define the problem of plan recognition as the problem of finding a goal and an optimal plan to achieve it while satisfying a set of observed actions. Ramırez and Geffner [34] provide a extensions that allow the inclusion of a fluent p (not only actions) in the sequence of observations by replacing them with *dummy* actions whose preconditions and effects are the same fluent p. They also discuss an extension where the sequence of observed actions can be partially ordered.

They assume a fully rational agent following an optimal plan to satisfy one or more elements of a set of possible goals. Unfortunately, this assumption does not hold for a human operator controlling a robot to perform ADLs, since the operator is not always acting rationally. For example, the operator might want to use the arm to pick up a bottle from a table top in order to make room for grasping a cup which was unreachable because of the bottle, after grasping, the operator will move the bottle and place it again on the table. The initial state for this scenario would be defined as p in (9), and it would also be the same goal state. An optimal plan that satisfies p starting from p is an empty action sequence, picking the bottle would eliminate p from the set of goals.

$$p = on(bottle,\ table) \tag{9}$$

Ramırez and Geffner [35] introduced a probabilistic approach for plan recognition using classical planners. The approach is similar to Ramırez and Geffner [34] but instead of eliminating solutions, they rank them according to their likelihood given the observations. The likelihoods of different goals are obtained using Bayes rule as shown in equation 10 where α is a normalizing constant, for goal G and an observation sequence O.

$$P(G|O) = \alpha\ P(O|G)\ P(G) \tag{10}$$

The likelihood $P(O|G)$ is calculated using the difference between the costs of a plan to achieve G while complying with O and a plan to achieve G while not complying with O.

Although this approach is robust to noise in the observation and deviation from optimal plans, multi-step-actions, such as picking a bottle by first moving a cup out of the way to make space for the manipulator need to be modeled to be used for recognizing long term plans. This might require geometric reasoning to be involved with the symbolic reasoning about the actions. Instead, the approach presented in this thesis deals with activity recognition one activity at a time.

4 The CARE Framework

The EDAN robot is introduced by the DLR as an assistance robot for people with tetraplegia. It consists of a DLR LWR arm with a DLR/HIT hand installed on a commercially available electric wheelchair. The wheelchair motion can be controlled by a joystick mounted on the arm rest or using EMG signals.

The LWR arm installed in EDAN is controlled using Cartesian impedance control [30] which offers a safe interaction with the environment during object manipulation [42]. It is teleoperated by the person using the chair to perform ADLs in a shared autonomy control paradigm through the Constraint-based Activity REpresentation (CARE) framework. The framework reasons about the user's input as well as the symbolic and geometric world state to enhance the user's commands by changing the manipulator trajectory to facilitate the performance of ADLs.

This section describes the main contributions of the thesis. The thesis focuses on using shared autonomy control for operating a robot arm using a low-dimensional input device to perform ADLs. This thesis presents an object-centric approach for action description in a shared autonomy paradigm. In this approach objects represent their interaction and manipulation actions using a combination of geometric and symbolic constraints. The CARE framework enhances a user's commands to follow the constraints defined in the actions of the different object classes.

The approach is demonstrated by performing three types of activities. The first activity is pouring the contents of a container into another. The second activity is wiping, such as the movements involved in cleaning a surface with a wiper. The third activity is opening a door or a drawer.

Figure 8: The DLR EDAN assistance robot.

Ways of implementing the pouring and door opening activities are discussed in [29]. In contrast to [29], this thesis presents a formal method of implementation for these activities. Moreover, It discusses fail scenario handling, such as what would happen if the planned door opening trajectory would cause the manipulator to go out of reach.

This section gives an overview of the system, describes the constraint-based activity representation and how the system handles this representation, and then describes the method used for intention inference.

4.1 System Overview

The CARE framework, shown in Figure 9, contains five main modules:

- The Object DataBase (ODB) introduced in [23].
- The world representation [23].
- The simulation environment (OpenRAVE [8]).
- The intention inference engine.
- The shared autonomy controller.

The object database contains information about the objects, such as dimensions, tool frame, upright orientation, and 3D models. The world representation contains information about the current symbolic and geometric state of the world. The ODB and world state were introduced and described in more detail in [23]. The intention inference module is used to reason about the current geometric and symbolic state of the world to find out the activity an operator is performing. The shared autonomy controller is the module that handles how the robot arm acts in response to the user input. This thesis contributes the latter two modules.

When the system receives input from the user, the intention inference module reasons about the symbolic state of the world to find the feasible actions. In case it finds multiple feasible actions, it chooses the closest object to the manipulator (or to the object held by the manipulator). After that, the geometric object information as well as object poses are obtained from the object database and world representation. The constraint-based activity representation, described in more detail in Section 4.2, is used to calculate an enhanced location and pose for the manipulator using the geometric information and the user's input.

A new pose is calculated as a response to each user command and is sent to the controller of the robot to get the corresponding configuration of the robot arm. This can be used to control the robot arm in the simulation or the real world.

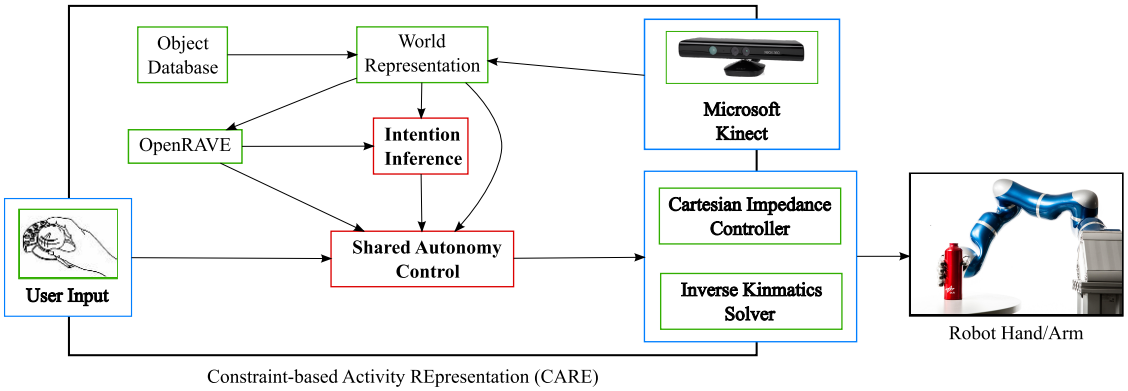


Figure 9: Overview of the CARE framework. The object database (ODB) contains information about object classes while the world representation contains concrete information about the objects in the environment. This thesis contributes the intention inference and the shared autonomy control modules (marked in red), which infer the user's activity and define how the arm reacts to the user input respectively. Interfaces to the world, marked in blue, are used to communicate information to the robot and receive the user input as well as information about the world. The two modules contributed by the thesis communicate with the other modules (marked in green) through wrappers and APIs.

4.2 Constraint-based Activity Representation

The CARE framework describes manipulation actions using constraint-based definitions similar to the constraint-based movement description language introduced by Bartels et al. [1]. Motion commands are represented using constraints defined using the geometric features of objects and a set of feature functions in [1]. This approach focuses on the success conditions of the task, leaving other aspects of the motion free for optimizing the movement. The CARE framework only focuses on satisfying the geometric constraints provided in the activity definition giving freedom to the user to change other aspects of the motion, such as approach direction and speed.

An activity in the CARE framework is divided into phases, as visualized in Figure 10, each of the phases (blue boxes) contains a number of constraints (marked in red). As seen in (11), an activity is formed of a list of phases p_i.

$$activity = [p_1, p_2, \ldots, p_n] \tag{11}$$

where each phase, p_i is a list of constraints as seen in (12)

$$p_i = [c_1, c_2, \ldots, c_m] \tag{12}$$

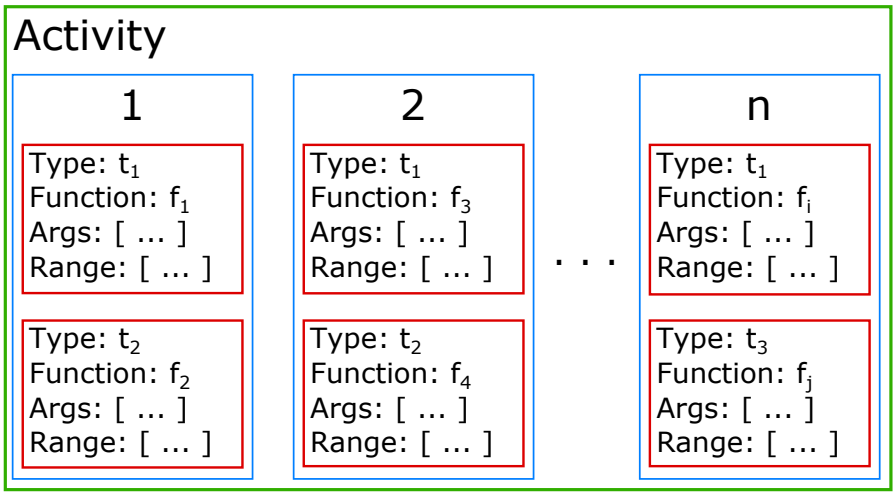


Figure 10: The general representation of an activity. Each activity is divided into phases, colored in blue, and each has a number of constraints, shown in red.

where c_j, as seen in (13) has a *type*, and a function f, and a list of arguments for f denoted *args*, it also has a list of numbers called *range*. The type of the constraints defines how the different attributes of the constraints are used by the shared autonomy control module and how the *rage* is interpreted for the different classes, as described in Section 4.3.

$$c_j = (type, f, args, range) \tag{13}$$

4.3 Constraint Types

The shared autonomy control module is responsible for enhancing the user's commands according to the constraints of the objects involved in the activity. This module queries the object for the constraints and calculates new frames for the manipulator that follow the activity constraints while following the user's commands.

Constraint are handled by this module according to their type. Different constraint types cause the manipulator and the end effector to have different behaviors. The module handles three different types of constraints:

- Geometric constraints.
- End effector constraints.
- Phase shift constraints.

The three constraint types use the definition explained in Section 4.2 and handle different types of motion as explained below.

4.3.1 Geometric Constraints

During the execution of an activity, manipulator motions have to be guided in order to ensure a successful task execution. This guidance is done by enhancing the user's commands to move the manipulator according to the activity definition. Geometric constraints are responsible for defining the properties of the motion that should be changed. They define changes of direction, speed, and type of motion (translational and rotational) necessary to perform an activity.

As shown in Figure 10, geometric constraints define a callable function f. This is the callback function that responds to a command coming from the operator. After an input is received from the user, f is called with *args*, which contains attributes of objects involved in an activity. *range* defines the range of motion, i.e. the axes of motion in the Cartesian space that are going to be affected by the transformation matrix returned by f as a response to the user's commands.

An example is shown in Figure 11, the activity of wiping a table with a sponge contains a single phase. The geometric constraint inside this phase is responsible for keeping the sponge in contact with the table. Its callback function projects the user's commanded motion onto the surface of the table and the *range* attribute indicates that it eliminates user commands that move the end effector along the z-axis (2 is the index of the dimension of Cartesian space controlling translation in z-axis).

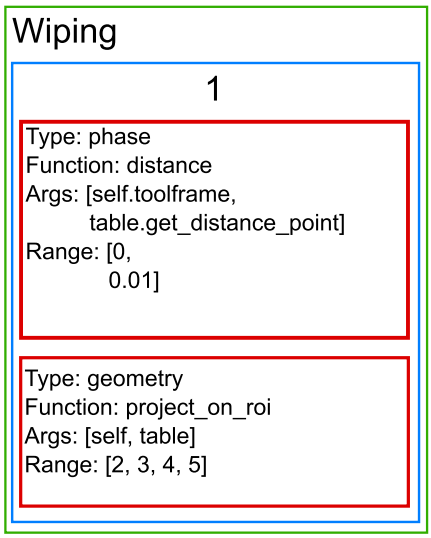


Figure 11: The constraints for wiping a table with a sponge.

If the object doesn't define any constraints for an activity, then the user commands are directly applied to the manipulator without any modification.

4.3.2 End Effector Constraints

Activities that require moving the fingers of the hand of the robot, such as picking an object or pressing buttons, need to send the new finger joint angles to the robot controller. To this end, the constraints include the end effector constraint class, the type responsible for sending end effector configuration to the controller of the robot when an activity requires this.

The callback functions f for this type of constraints return a list of robot operations (skills) to be executed by the manipulator such as `bind` or `release`, with their responding end effector configuration. The operations used here are a subset of the operations available for the action code snippets introduced by Leidner et al. [23] and used for the geometric planning. The CARE framework only includes the subset of these sequences that changes the configuration of the hand, while in [23] these sequences included more operators that are responsible for moving the manipulator and changing its pose which require planning time.

In the end effector constraint class, the list of arguments to the call back function, *args*, either contains the object whose grasp should be sent to the end effector or is empty, such as the case of resetting the grasp to release an object. The *range* list for this constraint class is always set to $[0, \ldots, 5]$ disabling all motion commands from the user till the grasp is applied.

4.3.3 Phase Shift Constraints

During an activity execution, different types of motion should be performed in different stages of the activity. Depending on which parts of an activity has been performed, a phase of the activity becomes active making its constraints responsible for the behavior of the arm. To decide when a phase ends and another one starts, the CARE framework uses the phase shift constraint class.

These constraints decide which phase of the activity is currently active using the test function f. The functions are called with the arguments provided by the *args* list, and decide whether or not the new phase should start. The functions f are distance functions and can be used to measure the distances between the end effector and an object or two objects.

For instance, the phase shift constraint in Figure 11 (upper red box) tests whether the sponge is within a $1cm$ distance from the surface of the table. Phase 1 is started when the sponge is within that distance, and the motion of the end effector is decided based on the returned transformation matrix from the respective geometric constraint (lower red box in the same figure).

In the pouring activity, as demonstrated by Figure 12, as the user moves a bottle towards a cup, the first phase is started at an appropriate distance between the bottle and the cup by tilting the bottle towards the cup (Figure 12a), the next phase starts when the toolframe of the bottle is above the rim of the cup (Figure 12b), the third phase is started when the pouring ends (when

(a) Phase 1 (b) Phase 2 (c) Phase 3

Figure 12: Different phases of pouring. The light brown bottles indicate the starting position of each phase and the darker ones indicate the respective end positions. Figure 12a shows the first step of the pouring activity, a bottle is rotated till its toolframe (rim) is about the cup. After that, the bottle is rotated around its toolframe till the rim is pointing towards the cup as seen in Figure 12b. At the end of the activity, it is rotated to an upright position as seen in Figure 12c.

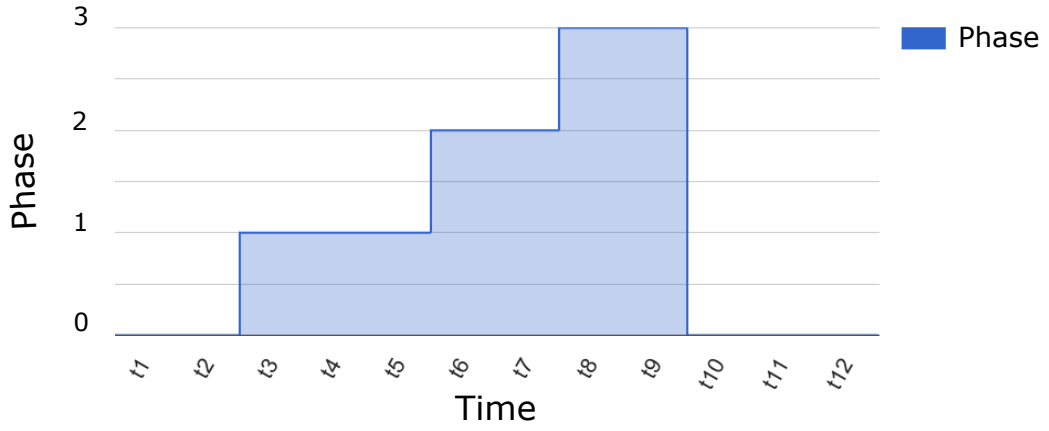


Figure 13: The currently active phase changes w.r.t time as the user sends commands to the robot. As seen in Figure 12, different types of motion are performed in the different phases, in this case, the rotation is executed around different points in Cartesian space depending on the currently active phase.

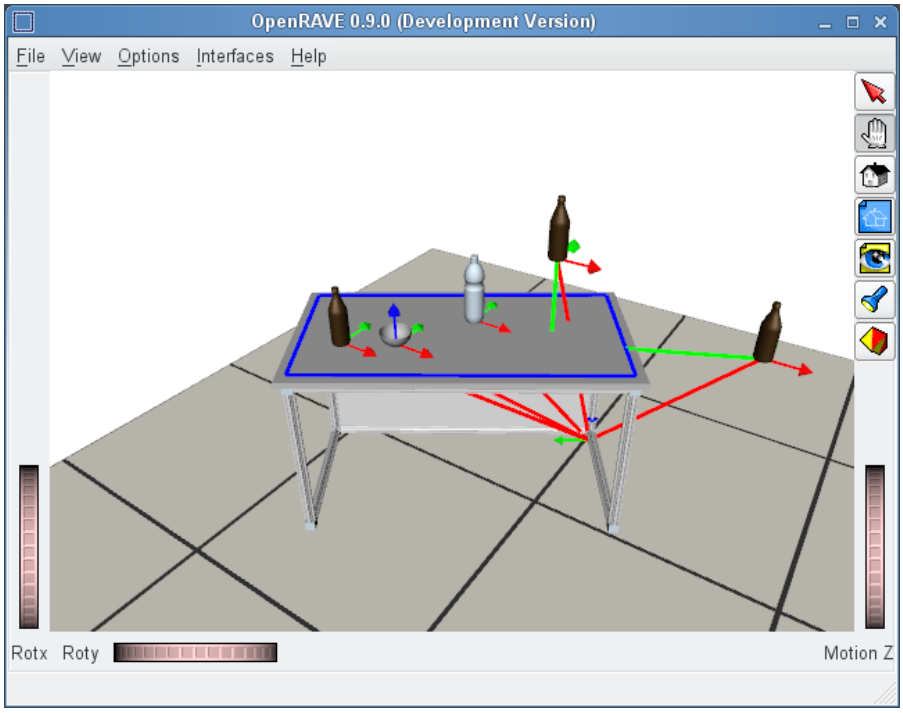


Figure 14: The distance between the table and different objects as calculated using the object frames (shown in red) and using the `get_distance_point` (shown in green). The Region Of Interest (ROI) of the table, shown in blue, is used for distance calculations.

the toolframe of the bottle is pointing inside the cup) to return it to an upright position (Figure 12c). Figure 13 shows how the phases change as the user moves the bottle towards the cup.

Deciding the currently active phase of an activity requires different distance functions between different object types. For instance, the distance between a bottle and a cup can be measured as the euclidean distance between their respective frames (or toolframes), while the distance between a bottle and a table cannot be measured in a similar manner. Because the frame of the table is defined to be on one of the lower corners and tables do not provide toolframe. For this reason, object classes in the object database provide a dynamic distance point function that returns a point in 3D space for measuring the distance between the two objects. The `get_distance_point` function of an object, used in the phase shift constraints of the table wiping activity as seen in Figure 11, takes as argument another object and returns a point depending on their types.

As seen in Figure 14, measuring the distances between objects using their frames leads to unrealistic distance measures (depicted by the red lines) since the frames of objects are defined to be on the bottom of the objects and in the case of the table, the frame is defined as one of the corners. The frame distances (red lines) can't be used to decide whether an object is close enough

to the table to start the release phase of a place activity, where the end effector constraints define a new configuration for the hand. Instead, distances are measured by projecting the frames of the objects on the Region Of Interest (ROI) of the table (marked by the blue rectangle) leading to more useful distances (shown in green).

4.4 Intention Inference

Without knowing what the user is doing, the system can not offer appropriate help. It wouldn't know how to move or rotate the manipulator in response to the commands of the user. To be able to assist the user correctly, the system needs to know the activity a user is performing and its parameters, e.g. the objects involved.

In the pouring activity, for example, if the system knows beforehand that the robot arm is holding a certain bottle and that the user is trying to pour its content in a certain cup, it can start applying the constraints defined for interaction between the bottle and the cup for the pouring activity. Unfortunately this isn't the case in the real world where most scenarios contain multiple objects that can afford more than one action. An environment could contain two (or more) cups in which pouring is possible while the manipulator is holding a bottle.

Using the symbolic planner from [23] and selecting the closest object to the manipulator to interact with, it is possible to find the symbolically feasible actions. However, this is slow and uncomfortable for the operator since it would have to be be performed every time an input is received from the user, which is time consuming.

Using the PDDL symbolic headers of the actions used in [23], the CARE framework provides an inference engine to find feasible activities. It invests startup time, creating a graph to represent the objects and possible actions in an environment, in order to save time later while performing a query to get the user's intention. Moreover, it uses information such as the contents of a bottle for reasoning such that a user doesn't unintentionally pour a liquid into an incorrect container, such as pouring water, coffee, or juice into a bowl of cereal or an already filled cup.

The nodes in the graph represent the objects in the world and the edges represent preconditions to actions that are represented by paths between the objects. The edges in the graph are split into two categories, enabled and disabled edges representing satisfied and unsatisfied preconditions respectively.

4.4.1 Graph Creation

The graph is created by parsing all the actions -feasible and infeasible alike- (from the ODB in [23]) of all objects in the environment and adding an edge for each of the preconditions of the actions of each object.

The set of nodes of the graph is initially created as a set containing a node mapped to each object. This initial set is traversed and the actions for each object are parsed. Paths start and end in

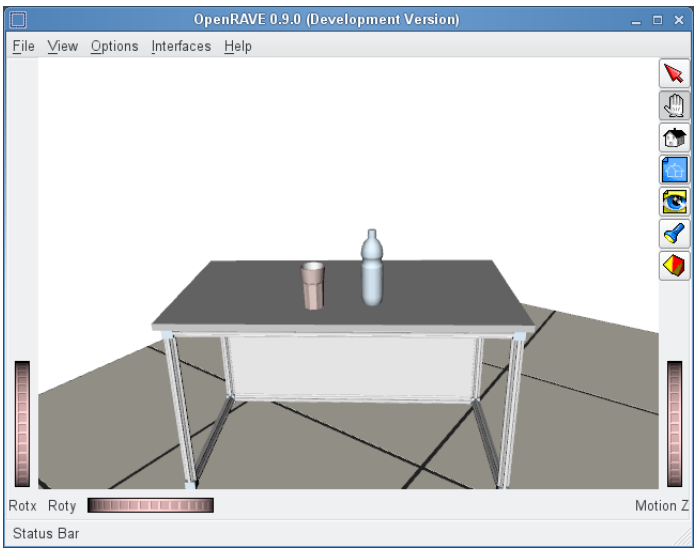


Figure 15: A rendered scene of a world containing a number of objects.

nodes representing objects, such that a path on the graph represents an interaction between the source and the target objects of the path. Hops on a path (edges) represent the preconditions of the action represented by that path.

An action with multiple preconditions is represented with a path containing multiple edges, the first edge starts from the node representing the object defining the action and an intermediate node is added to the set of nodes as a target to that edge. The path continues from the intermediate node by adding edges and intermediate nodes as needed until the goal object is reached.

For example, to represent the environment shown in Figure 15, the three objects (the bottle, the cup, and the desk) are represented using three nodes. The actions of each of the objects are parsed and their parameters are resolved.

```
'''
:parameters (?o - _object ?t - _tray  ?m - _manipulator)
:precondition (and (free ?m) (on ?o ?t))
:effect (and (bound ?o ?m) (not (free ?m)) (not (on ?o ?t)))
'''
```

Snippet 3: The symbolic headers involved in the action of picking an object on a tray. The preconditions are the free manipulator and that the object is on the tray. As a result of the action the object is bound to the manipulator, the manipulator is no longer free, and the object is not on the tray.

The parameters of the `pick` action, `o`, `t`, and `m`, seen in Snippet 3, can be resolved to `bottle`, `desk`,

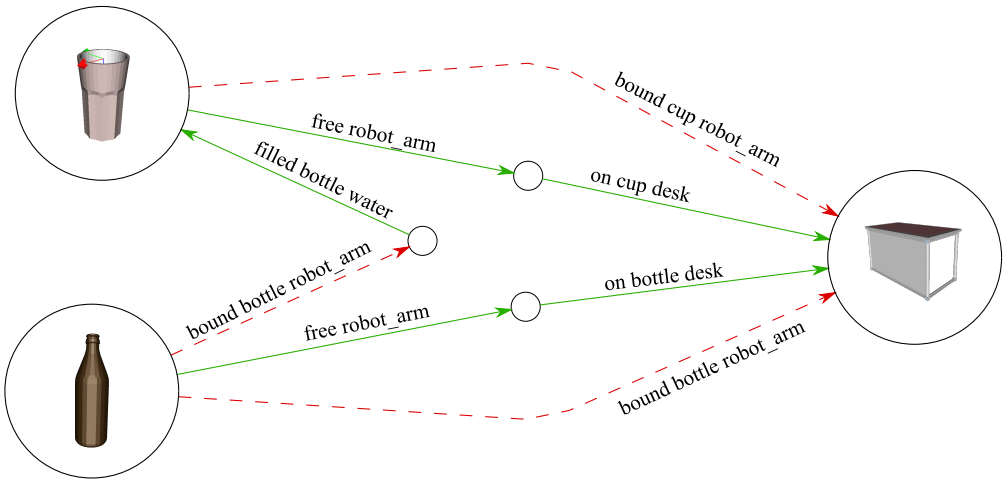


Figure 16: An Example of the graph-based intention inference approach. The figure shows the graph formed when the environment contains a bottle, a cup, and a desk. The graph contains five paths representing five different actions. Enabled edges are marked in the green arrows while the disabled ones are represented with the red dashed arrows.

and `robot_arm` respectively, meaning the bottle can be picked from the desk using the robot arm. This action will be represented using a path of length 2 connecting the bottle and desk nodes. One edge goes from the bottle to an intermediate node, and another from the intermediate node to the desk, as shown in Figure 16. Another path represents the action `place`, whose header is shown in Snippet 1, the path also connects the bottle node to the desk node and has length 1 since the action has a single precondition. Similarly, the pick and place actions of the cup are represented on the graph as two paths between the cup and the desk. The action of pouring from the bottle to the cup is also represented by a path of length 2 connecting the bottle to the cup as seen in Figure 16.

Preconditions are tested during the graph creation time, their edges are added to either the set of enabled or disabled edges according to their truth values. In the graph shown in Figure 16, the green edges are the enabled since they represent the satisfied conditions of the different actions, the red dashed edges are disabled since they represent preconditions that are not True, such as `bound bottle robot_arm`.

In order to save memory while storing the graph and time when updating the graph or querying for feasible actions, a dictionary is created to store the edges of the graph instead of storing them in a list. The keys of the dictionary are the preconditions representing edges as shown in Snippet 4.

```
Enabled Edges:
    free robot_arm:
        [ from: cup,
          to:   temp_5,
          for:  pick cup desk robot_arm,
          cond: free robot_arm ],
        [ from: bottle,
          to:   temp_6,
          for:  pick bottle desk robot_arm,
          cond: free robot_arm ],
        [ from: floor,
          to:   temp_8,
          for:  pick floor desk robot_arm,
          cond: free robot_arm ]
    on bottle desk:
        [ from: temp_6,
          to:   desk,
          for:  pick bottle desk robot_arm,
          cond: on bottle desk ]
Disabled Edges:
    bound bottle robot_arm:
        [ from: bottle,
          to:   desk,
          for:  place bottle desk robot_arm,
          cond: bound bottle robot_arm ],
        [ from: bottle,
          to:   temp_7,
          for:  pour bottle cup water robot_arm,
          cond: bound bottle robot_arm ]
```

Snippet 4: A subset of the graph edge storage representing the graph in Figure 16.

4.4.2 Querying and Updating the Graph

At run time, querying the graph for feasible actions is reduced to a graph connectivity problem. Starting from a node, all enabled paths are explored in a depth first search till they reach another object node or until no more enabled edges are available to continue this path. A path ending in an object node represents a feasible action. These paths only contain the enabled edges since the disabled ones are ignored because the preconditions they represent are not satisfied.

For instance, the graph shown in Figure 16 shows two feasible actions represented by the two green paths from the bottle to the desk and from the cup to the desk. Information on which action a path represents is stored as an attribute of the edge, as seen in Snippet 4 in the `for` attribute of the different edges.

After an activity such as picking the cup is completed, the graph is updated by moving the effects of the activity from the enabled dictionary to the disabled one. This way they will no longer be considered when querying the graph. This keeps the graph consistent with the current world state. Which means that parsing the actions as well as testing preconditions is no longer needed after the graph is created.

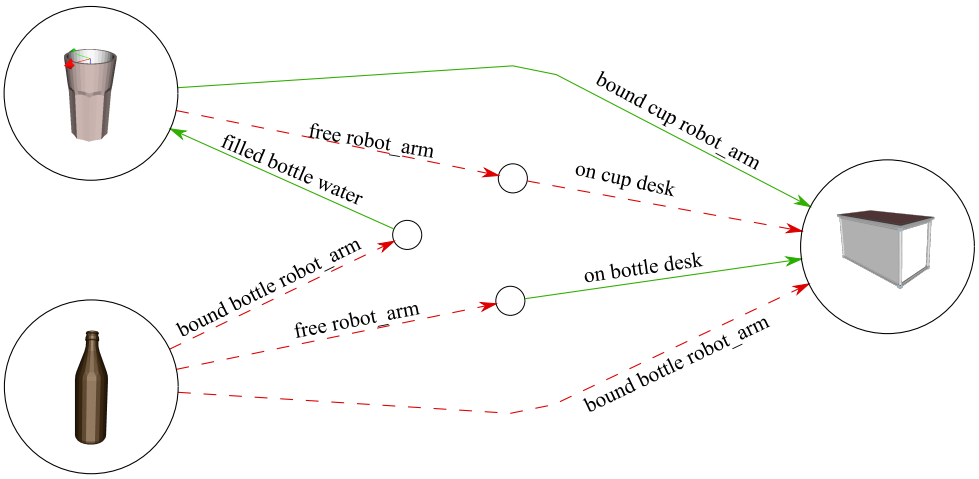


Figure 17: The updated graph resulting from the graph shown in Figre 16 after picking the cup with the end effector (robot hand).

For instance, if the hand picks the cup from the scene represented by the graph in Figure 16, the hand will not be free, all edges that represent this precondition will be disabled by moving them to the disabled list. The result is shown in Snippet 5 which shows a subset of the edges of the graph of Figure 17. The list of feasible actions is be obtained by following the connected paths (green) in the graph. The only feasible action after this update is placing the cup on the desk, this represented by a path of length 1 containing the edge `bound cup robot_arm`.

The graph also supports addition and removal of objects online after the graph creation. The addition of objects is done by parsing the new object's actions examining their preconditions. Edges to representing them are added to the graph in the enabled or disabled dictionaries according to whether the precondition is satisfied. Enabled and disabled edges are revised to update the graph according to the properties of the new object. For example, if the new object is added such that it is held by the end effector, the dictionary entry representing the condition `free robot_arm` is moved from enabled to disabled. Removing an object is done by removing its node from the graph and removing the paths connecting this node to others.

```
Enabled Edges:
    on bottle desk:
        [ from: temp_6,
          to:   desk,
          for:  pick bottle desk robot_arm,
          cond: on bottle desk ]
Disabled Edges:
    free robot_arm:
        [ from: cup,
          to:   temp_5,
          for:  pick cup desk robot_arm,
          cond: free robot_arm ],
        [ from: bottle,
          to:   temp_6,
          for:  pick bottle desk robot_arm,
          cond: free robot_arm ],
        [ from: floor,
          to:   temp_8,
          for:  pick floor desk robot_arm,
          cond: free robot_arm ]
    bound bottle robot_arm:
        [ from: bottle,
          to:   desk,
          for:  place bottle desk robot_arm,
          cond: bound bottle robot_arm ],
        [ from: bottle,
          to:   temp_7,
          for:  pour bottle cup water robot_arm,
          cond: bound bottle robot_arm ]
```

Snippet 5: The subset of graph edge storage representing the graph in Figure 17 which results from performing the activity of picking a cup starting from the world state represented by Figure 16.

4.4.3 Kinematic Reachability

Activities require an object or a location to be reachable by the robot. For instance, in the pouring activity, a frame has to be reachable where a bottle is placed above a cup and is tilted with an angle, as seen in Figure 18. Such preconditions can't simply be added as edges in our graph since they aren't well defined as there is an unlimited number of positions above a cup where the bottle can be tilted to perform the same activity.

The CARE framework provides an extension to deal with geometric preconditions. A cup (or any fillable container) provides one or more frames for the pouring activity, representing the toolframe of the bottle used for pouring, such as the position p_1 in Figure 18.

Instead of adding reachability of such frames to the bottle's pouring action directly, a precondition is added that says that a pouring frame from the target of the action (the cup) should be reachable as a precondition to pouring. Such edges are always enabled, and are only checked for geometric reachability when they are encountered while querying the graph for actions.

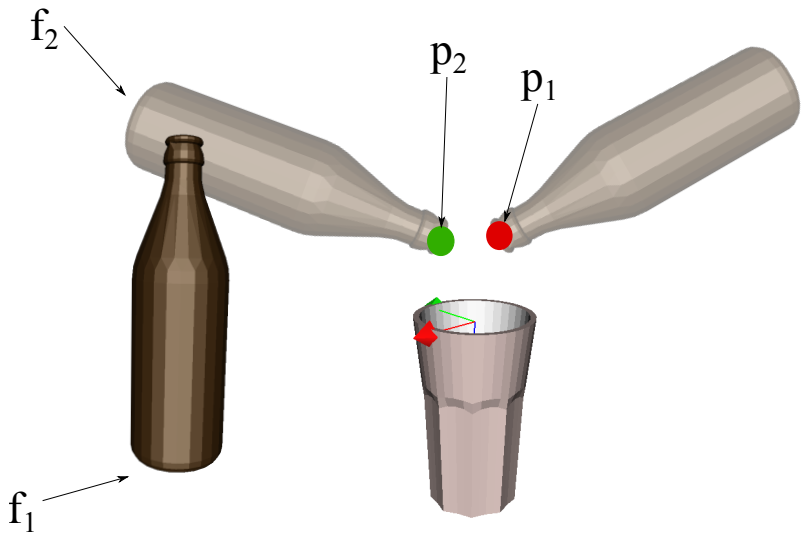


Figure 18: If the bottle is currently at position f_1 and the pouring frame defined in the cup class is p_1, but to check for reachability of pouring, the algorithm checks the reachability of f_2 that would place the bottle's toolframe at position p_2.

Checking the feasibility of reaching a pouring frame, consists of using the IK to check the kinematic reachability of that orientation at a point above the cup and between the toolframes of the bottle and the cup. Figure 18 shows an example predefined pouring frame p_1, to check the reachability of pouring when the bottle is at position f_1, the algorithm checks the reachability of the f_2 that would lead the bottle to a closer pouring frame p_2.

4.4.4 Algorithm Analysis

The intention inference module presented here uses symbolic as well as geometric information to infer the user's desired activity. The CARE framework uses this intention inference module to appropriately assist the user based on this activity. Using a combination of symbolic and geometric information, the framework is able to protect the user from engaging in an incorrect action as well as getting the arm to an undesirable joint configuration.

By adding preconditions to some actions of objects, it is possible to prevent the operator from engaging in actions that are going to fail. For example, adding a precondition to pouring which states that the bottle should be open protects the user from starting a pouring activity with a closed bottle. Moreover, adding a precondition to the action of opening a door which states that the final location of the door knob should be reachable, prevents the arm from getting into an undesirable joint configuration while trying to open the door.

Adding information about the contents of containers into the preconditions of the actions, pre-

vents an operator from accidentally performing unintended actions, such as pouring some liquid in an already filled cup.

A limitation of this approach for intention inference is that query runtime is dependent on the environment and the number of actions available. For n objects, each having a maximum of a actions where each action has p unique preconditions, the graph would have $n * a * p$ edges. At the worst case, all the $n * a * p$ edges would be enabled, causing the algorithm to traverse them all looking for symbolically feasible actions. In practice however, this can be handled by only considering objects that are reachable or that are within a specific distance from the robot since the robot's interactions are limited to objects within its reach.

4.5 Case Study

This section gives examples showing how different types of activities are defined in the CARE framework. It shows how the activities are divided into different constraints and phases and how to calculate the manipulator positions corresponding to a user's commands in the different phases of an activity.

4.5.1 Pick Activity

The ability to pick an object is crucial for everyday activities. It is important that a user is able to perform this activity confidently and comfortably. To this end, the pick activity is represented in the CARE framework in three phases. The phase transitions are seamless to provide an intuitive execution of the activity. In the CARE framework, the pick activity is divided into the following phases:

1. Adjusting end effector orientation.
2. Grasp path interpolation.
3. Apply grasp configuration.

The first phase of grasping, shown in Figure 19, is the phase where the end effector orientation is adjusted, i.e. the hand is rotated to point at the object. The second phase is where the user's commands are enhanced such that the end effector ends up at an appropriate grasping position if the user moves it towards the object. The third phase simply applies the grasp to the end effector, i.e. changes the configuration of the hand joints.

Adjusting End Effector Orientation

Grasps of an object can be obtained either from a predefined graspset, or using a computer vision based approach to find possible grasps for arbitrary objects such as [19, 22, 36], the CARE framework uses the predefined graspsets of objects as follows.

The predefined graspset of an object, such as the wiper (shown in Figure 20a) and the drawer (shown in Figure 29), can be obtained from the object database. Normally, the graspset of an object contains multiple grasps. The closest grasp to the hand is selected from the object's graspset and used to enhance the user's commands to guide the performance of the pick action. If the selected grasp frame g, shown in (14), where the g_{ij} components represent the orientation and $\begin{bmatrix} g_x g_y g_z \end{bmatrix}^T$ is the position of the end effector relative to the object frame f.

$$g = \left[\begin{array}{cccc} g_{11} & g_{12} & g_{13} & g_x \\ g_{21} & g_{22} & g_{23} & g_y \\ g_{31} & g_{32} & g_{33} & g_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{14}$$

and the commanded end effector frame, e has the form

$$e = \left[\begin{array}{cccc} e_{11} & e_{12} & e_{13} & e_x \\ e_{21} & e_{22} & e_{23} & e_y \\ e_{31} & e_{32} & e_{33} & e_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{15}$$

the rotation component of the hand frame can simply be replaced by that of g after converting it to the world frame, as seen in (16)

$$g' = f.g = \left[\begin{array}{cccc} g'_{11} & g'_{12} & g'_{13} & g'_x \\ g'_{21} & g'_{22} & g'_{23} & g'_y \\ g'_{31} & g'_{32} & g'_{33} & g'_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{16}$$

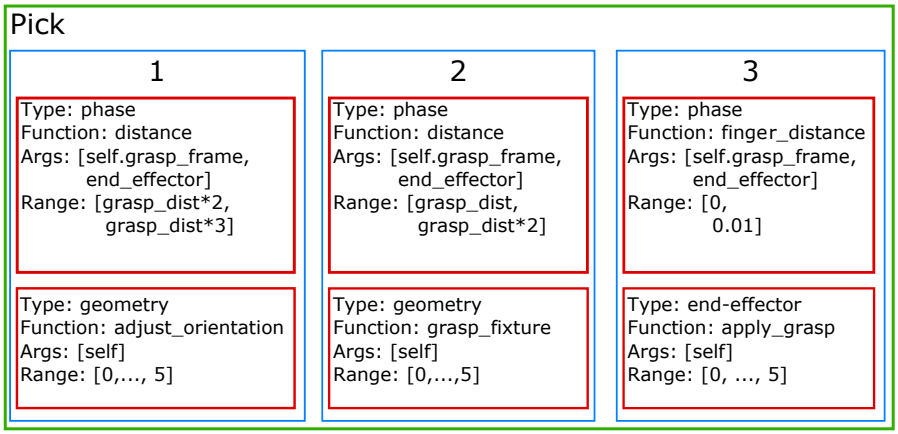
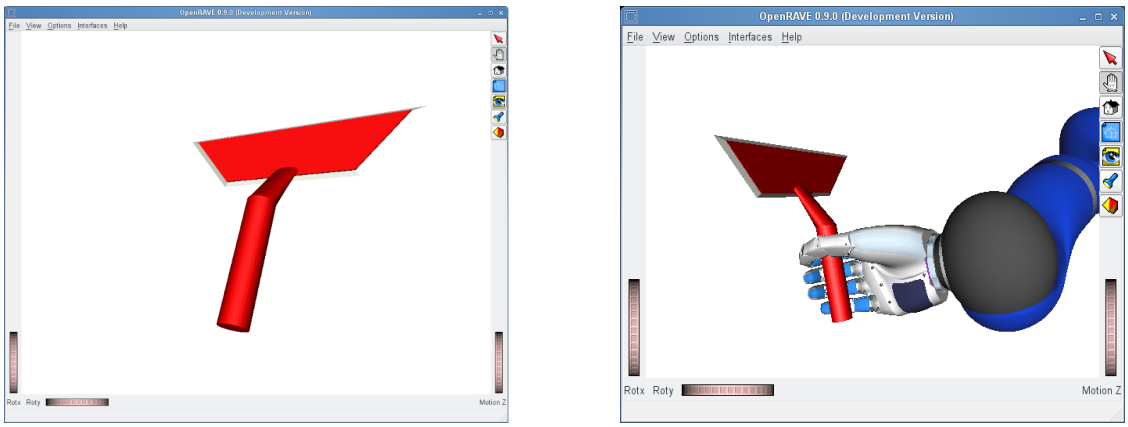


Figure 19: The activity of picking an object described using the constraint-based representation.



(a) (b)

Figure 20: Figure 20a shows a model of a window wiper from the ODB, Figure 20b shows the wiper grasped by the robot hand from the only predefined grasp position.

resulting in a new end effector frame e' in (17)

$$e' = \left[\begin{array}{ccc:c} g'_{11} & g'_{12} & g'_{13} & e_x \\ g'_{21} & g'_{22} & g'_{23} & e_y \\ g'_{31} & g'_{32} & g'_{33} & e_z \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{17}$$

The need for this can be seen by considering the example of grasping the wiper from Figure 20a, which can only be used if it's grasped from a certain direction as shown in Figure 20b.

Using only the predefined grasps to grasp a symmetric object would significantly reduce the flexibility and amount of user control involved in some actions. The objects in the object database might have only few predefined grasps which might be on the other side of the object or in an unreachable position.

For example, in Figure 21, the end effector would have to move from position e_1 to the closest predefined grasp g_1 in order to grasp the bottle. But due to the symmetry of the bottle around the z-axis, it is more intuitive if the hand is rotated to grasp the bottle from position e_2.

Another limitation for directly using the predefined grasps can be observed by examining Figure 22, starting from the initial configuration (position 1 on right side of the bottle) the grasp frame indicated by the coordinate frame (position 2 to the left of the bottle) may very likely be unreachable, since moving the hand to that location might cause the wrist joint to reach a joint limit.

The symmetry of an object can be exploited to overcome these limitations. An axis-symmetric object (an object that is symmetric around an axis) that has a grasp at distance d can be grasped

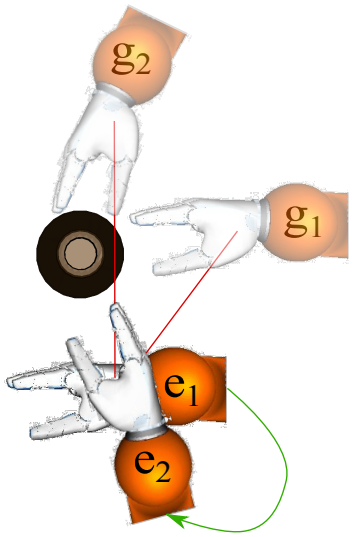


Figure 21: A bottle which only offers two grasps, g_1 and g_2, in its graspset would cause the end effector to move along the red lines to be able to pick the bottle. It would be more intuitive to rotate the hand along the green curve from position e_1 to position e_2.

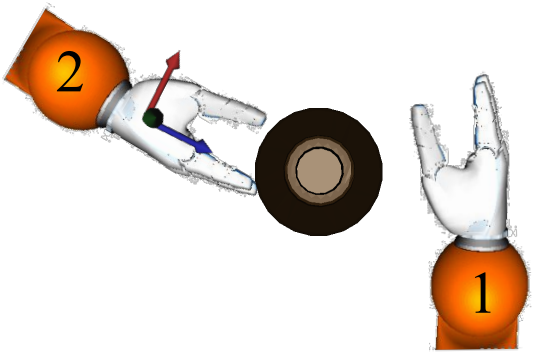


Figure 22: A bottle which only offers one grasp in its graspset, the end effector is at position 1 is on one side of the bottle while the grasp provided, position 2, is on the other side. To use the predefined grasp, the hand has to move around the bottle, which is redundant since grasping the bottle from any direction would lead to the same result due to the symmetry of the bottle.

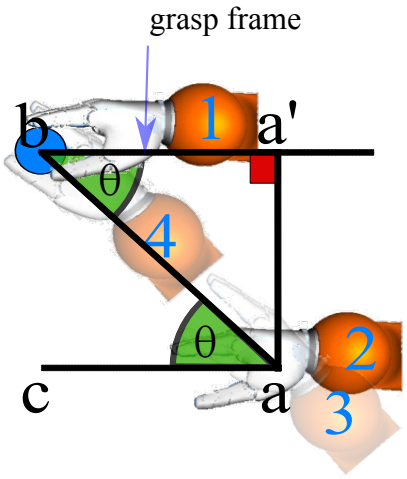


Figure 23: Predefined and approximate grasp frames for the bottle. The predefined grasp frame (hand position 1) is at distance d from the toolframe of the bottle, the approximate grasp frame is at distance d from the toolframe from the other side of the bottle. The approximate grasp frame is rotated so that the manipulator is pointed at the bottle.

from a circle in 3D space with radius d centered on the frame of the object if the circle is on a plane that has the relative height of the predefined grasp. This only holds if the hand is rotated such that it still points to the object, therefore the grasp frame has to be rotated to keep the angle between the object and its original grasp frame. The distance and angle are combined into a homogeneous transformation matrix to obtain an approximation for a grasp frame as seen in Figure 23.

In Figure 23 hand position number 1 is a predefined grasp for the object and position number 2 is the current -arbitrary- end effector position. If the predefined grasp frame is translated to position 2 it will point in a direction parallel to that of position 1. If a line ab is extended between position 2 and the object location, it will form alternating angles (θ, highlighted in green) between the two parallel directions, therefore rotating the grasp frame by angle θ will result in end effector frame 3, pointing towards the object.

Angle θ can be calculated as shown in (18), and depending on the quadrant where position 2 lies, a rotation angle ϕ can be found using θ and the relationships between the angles formed when the two parallel lines $a'b$ and ac, are intersected by a transversal line ab demonstrated in Figure 24, where a' is the projection of a on the line between the object and the predefined grasp frame.

$$\theta = tan^{-1}(\frac{aa'}{a'b}) \tag{18}$$

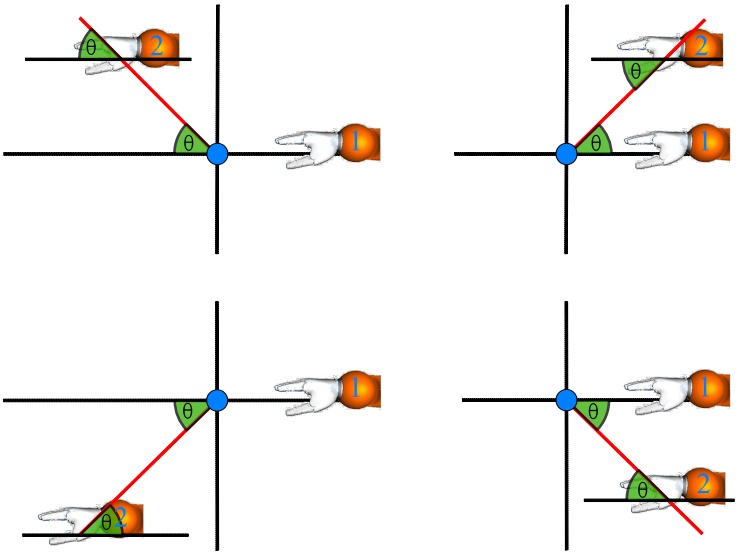


Figure 24: Relation between the calculated θ from (18) and the actual rotation angle depends on the quadrant where the grasping position lies. The figure shows how to get the rotation angle depending on different possible location for manipulator position 2 mentioned in Figure 23.

After that, a rotation matrix by ϕ around the y-axis of the end effector is created, as seen in (19), and applied to the grasp frame (20) and then the frame is translated by $\left[e_x, e_y, e_z\right]^T$ (the current end effector position) to obtain the frame for position 3 in Figure 23 as seen in (21).

$$r(\phi) = \left[\begin{array}{ccc:c} cos\phi & 0 & sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -sin\phi & 0 & cos\phi & 0 \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{19}$$

$$g' = g \cdot r(\phi) \tag{20}$$

$$g'' = \left[\begin{array}{ccc:c} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \cdot g' \tag{21}$$

Grasp Path Interpolation

For the second phase of the pick action, the end effector is properly oriented for a grasp (as a result of the previous phase), therefore it only needs to be moved to a position that enables a successful grasp. To this end, user's commands are projected on a virtual straight line connecting the end effector frame and a frame close enough to the object to grasp it, which is henceforth referred to as the interpolated grasp frame.

In this phase, the general case, for objects such as a wiper or a drawer is straightforward; the interpolated grasp frame is one selected from the predefined graspset.

For the symmetric object case, the interpolated grasp frame is placed at the same distance from the object as the predefined grasp frame, which lies on the line between the current end effector frame and the object (position 4 in Figure 23).

For an object placed at $p(f) = \left[f_x, f_y, f_z\right]^T$, if the end effector were placed at $p(e) = \left[e_x, e_y, e_z\right]^T$ (obtained in the first phase from (21)), the distances between end effector frame and the object as well as that between the selected predefined grasp frame and the object can be calculated as shown in (22) and (23) respectively, where p is a projection function that returns the position of a frame.

$$d_{e,f} = \sqrt{(e_x - f_x)^2 + (e_y - f_y)^2} \tag{22}$$

$$d_{g,f} = \sqrt{(g_x - f_x)^2 + (e_y - f_y)^2} \tag{23}$$

the ratio between the two distances is calculated as r

$$r = \frac{d_{e,f}}{d_{g,f}} \tag{24}$$

which is used to find the new x and y coordinates (g'''_x and g'''_y respectively) of the interpolated grasp frame

$$g'''_x = r * e_x + (1 - r) * g_x \tag{25}$$

$$g'''_y = r * e_y + (1 - r) * g_y \tag{26}$$

which are used to get the location of the interpolated grasp frame as

$$p(g''') = \begin{bmatrix} g'''_x \\ g'''_y \\ g_z \end{bmatrix} \tag{27}$$

After that, a virtual fixture is used to lead the hand to the interpolated frame by drawing a virtual line between the current manipulator position and the interpolated grasp frame and projecting the operator's commands on this line.

As seen in Equations (28) to (30), and the manipulator frame is translated to the location that results from this projection as seen in (31).

First the vector $\vec{v}$ from the current end effector frame to the interpolated grasp frame is calculated

$$\vec{v} = p(e) - p(g''') \tag{28}$$

and similarly, $\vec{e}$ from the current frame to the user commanded frame,

$$\vec{e} = p(e) - p(e') \tag{29}$$

then $\vec{e}$ is projected on $\vec{v}$

$$e'_{\|v} = p(e) + (\vec{e} \cdot \|v\|) * \|v\| \tag{30}$$

which can be used as the position component in the resulting frame without changing the orientation, since it's already adjusted from the previous phase

$$e' = \left[\begin{array}{ccc:c} e'_{11} & e'_{12} & e'_{13} & \\ e'_{21} & e'_{22} & e'_{23} & e'_{\|v} \\ e'_{31} & e'_{32} & e'_{33} & \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{31}$$

Figure 25 shows the result of projecting the commands on the grasping path. The user's commands applied at different locations are represented by the red lines, and the actual path taken by the manipulator is represented with the single green line. The figure shows how the green path follows the user's commands (the right side of the path) until the hand is close enough to the handle of the drawer to trigger the constraint of the virtual fixture for the pick action. After that all the commands in the direction of the handle are projected on the straight line leading to the interpolated grasping location. Commands leading the hand away from the handle aren't projected on the virtual fixture allowing the user to cancel the activity by moving away from the drawer.

Grasp Configuration

Using this virtual fixture to perform the activity is not sufficient since the manipulator may skip the interpolated grasp location by a few millimeters depending on the velocity commands from the user, causing the grasp to be inaccurate, to collide with the object (pushing it away), or causing the robot to believe that the object is bound to the manipulator while this is not the case.

To overcome this, the third phase of the picking activity, containing the end effector constraints, uses the `finger_distance` function. This function measures two distances, the distance between the frame of the object and the frame of the thumb, and the distance between the frame of the object and the frame of the forefinger. These distances are compared to the corresponding distances of the predefined grasp frame and the phase is only started when they are within range.

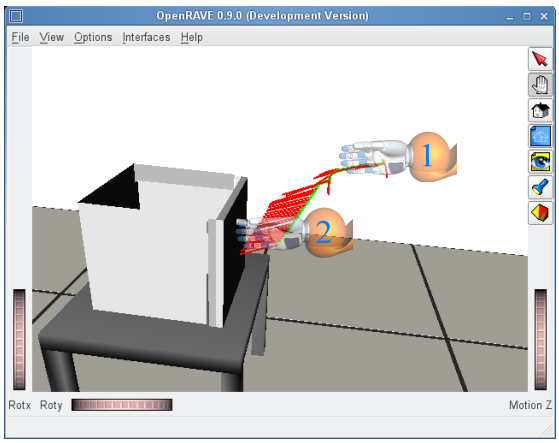


Figure 25: The user's commands at different positions (marked in red) projected on the grasping path to yield the actual path followed by the manipulator (marked in green) starting from position 1 and ending in position 2 where it is able to grasp the handle of the drawer.

The transformation between a finger at position t and the end effector location, e is measured as follows

$$d_{e,t} = e^{-1} \cdot t \tag{32}$$

which is then used to find the location of the finger when the end effector is at some frame $\dot{e}$

$$t' = \dot{e} \cdot d_{e,t} \tag{33}$$

after that the euclidean distance is calculated as shown in equation (34). This is done for both the forefinger and the thumb and the grasp is triggered within a certain threshold.

$$d'_{e,t} = \sqrt{(f_x - t_x)^2 + (f_y - t_y)^2 + (f_z - t_z)^2} \tag{34}$$

4.5.2 Pouring

Pouring is a basic activity involved in many compound everyday activities such as preparing a drink or preparing breakfast. This activity needs to be accurate and at the same time the user needs to feel that the execution is natural and comfortable.

To make the pouring activity intuitive for the operator, the pouring activity implementation exhibits the behavior shown in Figure 12 by using the description of Figure 26, dividing the activity into three phases as follows:

1. Rotating the bottle around the bottle frame (bottom).

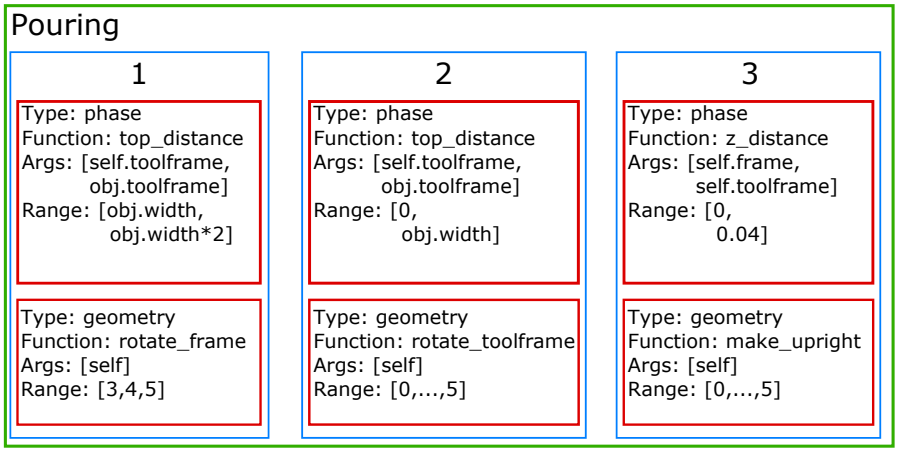


Figure 26: The pouring activity representation using constraints. The activity is divided into three phases, In the first phase, the bottle is rotated around its frame (base). In the second phase, it is rotated around its toolframe (rim). In third phase, it is rotated to an upright position.

2. Rotating the bottle around its toolframe (the bottle rim).

3. Rotating the bottle to an upright position after pouring.

More phases can be added to keep the bottle in upright position at all times until the rotation of the first phase starts, but they are not needed in this setup since the environment has only one agent making changes and all rotations of the SpaceMouse (or any other input device) were disabled by only considering a subset of the commands. Moreover, extra phases can be added to limit pouring from certain angles, by adding virtual fixtures around the target container and projecting the user commands on them, similar to the second phase of the pick activity.

Figure 26 shows how this activity is represented in the CARE framework. The activity is divided into three phases, each contains a phase shift constraint and a single geometric constraint. The first one rotates the bottle around its frame as shown in Figure 12a, the second rotates it around its toolframe as seen in Figure 12b the third phase returns it to an upright position as seen in Figure 12c to prepare it for being moved away.

Snippet 6 shows the preconditions and effects of the pouring action, the preconditions don't mention whether or not the bottle is open, this is because all bottles are assumed to be open since the environment setup has only one manipulator which means it can't open bottles.

This assumption doesn't cause any loss of generality, because this precondition can be added to the action of pouring, as mentioned in Section 4.4, with predicates that say whether a bottle is open in the world state representation, such preconditions will be seamlessly handled by the intention inference rendering actions of pouring from unopened containers infeasible.

```
'''
:parameters (?src - _bottle ?goal - _container
                ?l - _content ?m - _manipulator)
:precondition (and (bound ?src ?m)
                (not (filled ?goal ?l)) (filled ?src ?l))
:effect (and (filled ?goal ?l) (not (filled ?src ?l)))
'''
```

Snippet 6: The symbolic headers of the pouring action. The preconditions are that the object (bottle) is bound to manipulator and that it's full, and that the target container is not full. As a result of the action the source container is not full, and the target container is full.

For the rotation of the first pouring phase, the rotation angle θ is found by scaling the commands coming from the user, that way the user will have control on the pouring and can stop it at any time.

For the axis of rotation, a line $\vec{r}$, is used which is perpendicular to a plane defined by the bottle frame f_b, the cup frame f_c, and the cup toolframe t_c and intersecting it at the bottle frame.

The two lines $\vec{l_1}$ and $\vec{l_2}$ representing the axis of the cup and the line between the frames of the cup and the bottle are calculated using (35) and (36) respectively.

$$\vec{l_1} = t_c - f_c \tag{35}$$

$$\vec{l_2} = t_c - f_b \tag{36}$$

The rotation axis, $\vec{r}$ from (37) is the cross product of $\vec{l_1}$ and $\vec{l_2}$

$$\vec{r} = l_1 \times l_2 \tag{37}$$

and is used to calculate the rotation matrix $m(\theta, [x, y, z]^T)$ shown in (38)[1], where m is a function that returns a rotation matrix around $[x, y, z]^T$ by angle θ.

$$m(\theta, \begin{bmatrix} x \\ y \\ z \end{bmatrix}) = \left[\begin{array}{ccc:c} c\theta + x^2 * (1 - c\theta) & x * y * (1 - c\theta) - z * s\theta & x * z * (1 - c\theta) + y * s\theta & 0 \\ x * y * (1 - c\theta) + z * s\theta & c\theta + y^2 * (1 - c\theta) & y * z * (1 - c\theta) - z * s\theta & 0 \\ x * z * (1 - c\theta) - y * s\theta & y * z * (1 - c\theta) + x * s\theta & c\theta + z^2 * (1 - c\theta) & 0 \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{38}$$

The frame of the bottle is rotated using (39)

$$f_b' = f_b \cdot m(\theta, \begin{bmatrix} x \\ y \\ z \end{bmatrix}) \tag{39}$$

[1] Where $c\theta$ and $s\theta$ are $cos(\theta)$ and $sin(\theta)$ respectively.

The second phase is similar to the first apart from using a different rotation axis $[x', y', z']^T$, which is perpendicular to the plane defined by the bottle toolframe t_b, the cup frame f_c, and the cup toolframe t_c.

After creating the new rotation matrix $m(\theta, [x', y', z']^T)$, the transformation is applied to the toolframe of the bottle to find the new the new toolframe t'_b

$$t'_b = t_b \cdot m(\theta, \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}) \tag{40}$$

which is used to find the corresponding bottle frame f'_b using (41) where $\bar{t_b}$ is the bottle toolframe relative to its frame.

$$f'_b = t'_b \cdot \bar{t_b}^{-1} \tag{41}$$

The last phase for the pouring is started when the bottle's frame is higher than its toolframe, as shown in Figure 12c, the rotation component of the bottle's frame is changed to the identity matrix and the action is marked as done triggering an update of the graph in the intention inference module. After that the arm controller handles the transition between the current -tilted- pose to the upright position. A smoother transformation is possible by either rotating again around the bottle's frame in the opposite direction or inverting the previous two phases by rotating around the bottle toolframe with angle $-\theta$ then around the bottle frame.

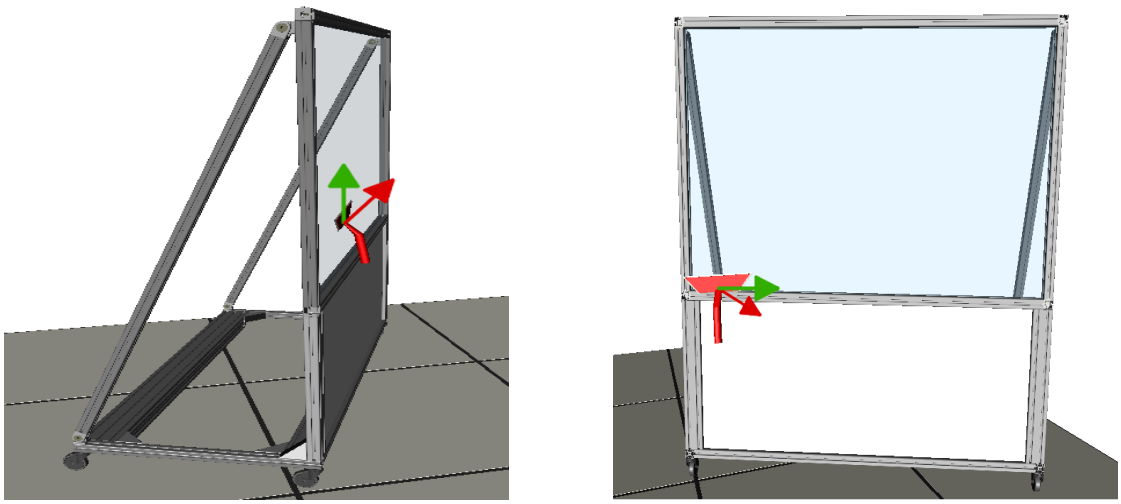
4.5.3 Wiping

Contact guided motion is involved in complex everyday activities, such as cleaning or scratching oneself. Examination of complex tasks that involve human body parts at the moment is not possible due to safety restrictions as well as difficulty in simulating activities that include these parts.

For activities such as cleaning a surface with a sponge, the user's commands are projected on the surface. Constraints are placed on the distance between the sponge or wiper held in the manipulator and the plane of the surface. The user is allowed to break contact with the surface by giving commands that move away from the surface. If the intended position was going away, contact with the surface can be broken.

This is done in a single phase whose constraints are shown in Figure 28, the callback function `correct_roi` performs three steps:

1. Apply motion to the wiper toolframe.
2. Project the new toolframe on the surface of the window.
3. Check if it's inside the ROI of the window.



(a) (b)

Figure 27: Some of the motions involved in wiping a window. The red arrows indicate the direction of the user's command and the green arrows indicate the direction of actual motion. Figure 27a shows the behavior when the operator sends commands that would cause the wiper to go away from the window, while figure 27b shows what should happen if the wiper is accidentally crossing the frame of the window.

4. Calculate the corresponding wiper frame.

Figure 27 shows how different commands are handled while cleaning a window with a wiper, the user's motion commands (shown by the red arrows) are projected on the glass and kept inside the ROI of the window. Figure 28 shows the constraints restricting the wiper from unintentionally leaving the region of interest of the window. The geometric constraint makes a call for the callback function `correct_roi` when the user moves the wiper placing its toolframe $1cm$ away from the plane of the window. This callback function keeps the toolframe of the wiper in contact with the plane at all times.

In the ODB, the window class defines a region of interest using three points in 3D space p_1, p_2, and p_3. The plane of the surface is defined by two lines connecting the three points forming the ROI. The two lines are $\vec{u}$ and $\vec{v}$

$$\vec{u} = p_2 - p_1 \tag{42}$$

$$\vec{v} = p_3 - p_1 \tag{43}$$

The normal vector to that plane is calculated using the cross product

$$\vec{n} = \vec{u} \times \vec{v} \tag{44}$$

and the user's intended toolframe t projected on the plane resulting in t' calculated in (45)

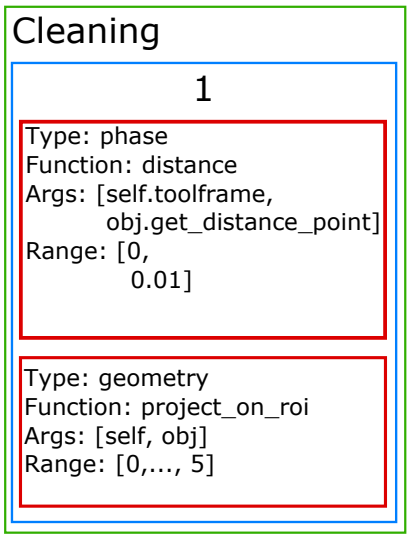


Figure 28: The constraint based representation for the window cleaning activity.

$$t' = t - (t - p_1 \cdot \vec{n}) * n \tag{45}$$

Following the outline of the window is achieved by projecting the plane of the window and t' onto the x-z plane, to get t'' and p'_i, and then checking if t'' is inside the rectangle formed by p'_i. If it's not, then the different components of the user's commands t are tested and the component that would keep t'' and consequently the wiper's toolframe inside the region of interest of the window is applied as seen in Figure 27b.

4.5.4 Opening

Path following activities such as opening a drawer or opening a sliding or rotating door are needed everyday specially when the user is operating the robot arm on the wheel chair which provides mobility. For this thesis these tasks were tested by opening the drawer of a night stand.

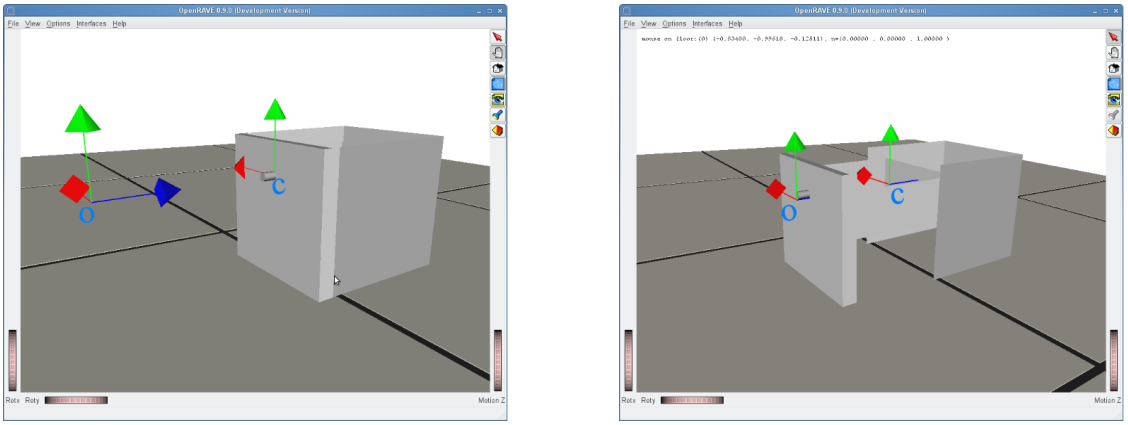
The drawer class defines, among other information, two points o and c representing the location of the handle in opened and closed positions respectively relative to the original drawer frame, as shown in Figure 29. Figure 30 shows the trajectory for opening a drawer calculated using the frames o and c.

The opening action has the preconditions defined in Snippet 7. As shown in Snippet 7, the effects say that the drawer is no longer bound to the manipulator and that the manipulator is free, this is due to the integration of the release of the drawer with the opening (and closing) actions of the drawer, which means that the operator will not be able to release the drawer when it's half opened (or half closed) to avoid any inconsistencies between the geometric and symbolic world states.

```
'''
:parameters (?o - _object ?b - ikea_night_stand_drawer ?m - _manipulator)
:precondition (and (bound ?o ?m) (not (open ?o)))
:effect (and (open ?o) (not (bound ?o ?m)) (free ?m))
'''
```

Snippet 7: The symbolic headers of the drawer-opening action. The preconditions are that the drawer is bound to manipulator and that the object is not open. As a result of the action the drawer is no longer bound bound to the manipulator, and that the manipulator is free.

The opening and closing actions, as shown in shown in Figure 31, consist of two phases. The first a virtual fixture projecting the operator's commands on the line between the opening and closing frames, seen in Figure 30 defined in the drawer class. The projection is applied using the same technique used for the projection phase (phase two) of pick action described in Section 4.5.1.



(a) (b)

Figure 29: A drawer object from the ODB showing the frames of the handle in case the drawer is closed (Figure 29a) and opened (Figure 29b).

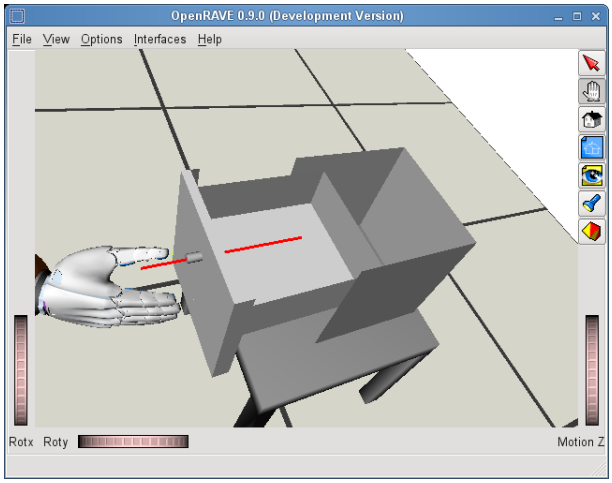


Figure 30: The figure shows the trajectory that should be taken to open the drawer highlighted in red. The start and end points of this line are defined by the nightstand class in the ODB.

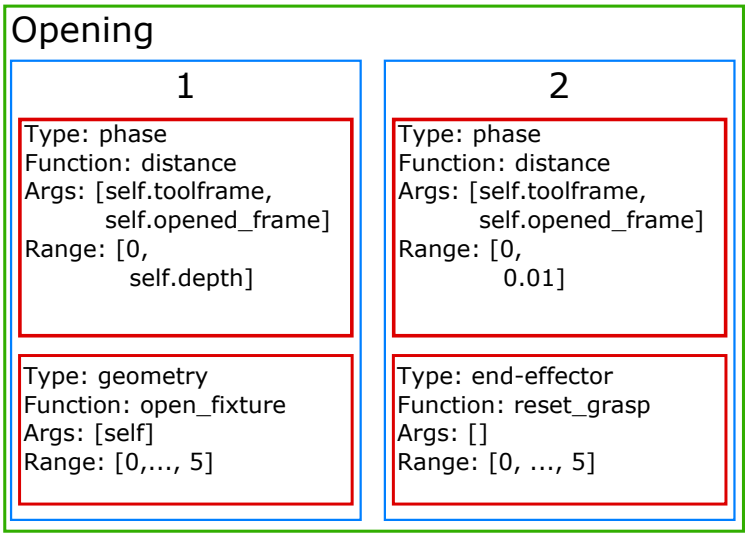


Figure 31: The constraint based representation for the opening a drawer of a night sand.

The second phase contains an end effector constraint responsible for releasing the drawer handle by setting the configuration of the hand to its default. The end of the opening action is declared triggering an update to the intention inference graph.

Using this activity definition for opening a sliding door is straight forward since the trajectory can be provided by the door class in the ODB. This can also be applied to a rotating door after a slight modification, by projecting the commands on the path formed by opening the door; a circle centered on the door's frame and lying on the horizontal frame of the door knob (toolframe).

5 Evaluation

In the future, the assistance robot, EDAN shall be teleoperated by people with tetraplegia to perform ADLs. The CARE framework will be used to determine the behavior of EDAN in response to user commands.

To demonstrate and evaluate the CARE framework, three ADLs were used; pouring, wiping, and drawer opening. These ADLs were chosen because they consist of different types of motion (translational and rotational) and are involved in different everyday situations. The implementation details involved in these ADLs are explained in Section 4.

5.1 Setup

The experiments were conducted using OpenRAVE [8] simulation environment. The experiments used a German Aerospace Center/Harbin Institute of Technology (DLR/HIT) hand [24] mounted on a DLR LWR arm [3]. The setup shown in Figure 32, shows the simulation of the arm/hand system mounted on a platform and kept at an angle for better reachability and ease of use. A SpaceMouse [14] is used to emulate a low-dimensional BCI for input during the work of this thesis.

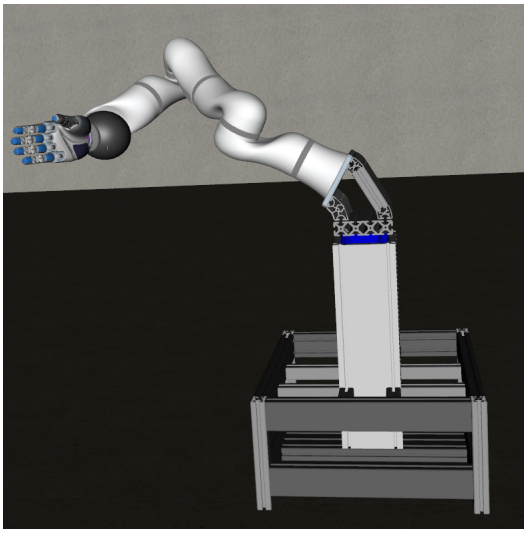


Figure 32: Rendered image of the DLR LWR arm setup used during the experiments. The platform keeps the arm/hand system at an angle increasing the reachability of the arm.

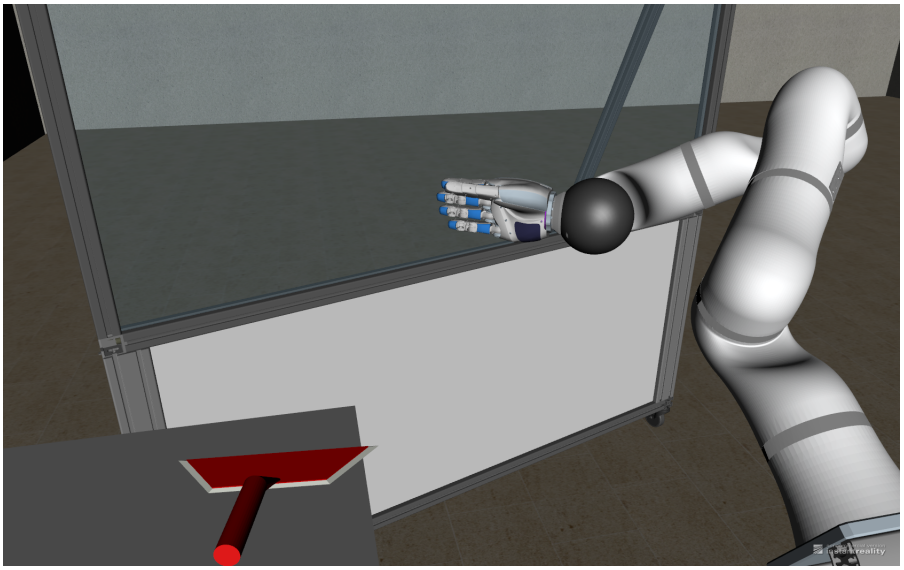


Figure 33: One of the rendered scenes shown to experiment participants.

5.2 Participants

The participants of the experiments are between 20 and 30 years old. The participants had no prior experience in robot teleoperation, three participants had limited experience using a SpaceMouse/ joystic.

5.3 Experiments

During the experiments, the users were shown a rendered scene of the LWR arm from the point of view of a person sitting on the chair mounted with the EDAN robot as seen in Figure 33.

The users faced different scenarios that a person may face in their everyday life while performing ADLs. Test scenarios involved more than one activity and overlapped in the pick and place actions. This allowed testing pick and place actions for different types of objects (symmetric and asymmetric objects) as mentioned in Section 4.5.1. The tests also examined grasping different objects from different directions.

Since the experiment participants were not used to robot teleoperation or using a SpaceMouse, at the beginning of each experiment session, the participant was given time ($\approx$ 2 minutes) to freely move the robot arm using the direct commands coming from the SpaceMouse to get used to the setup. After that, a pick and place task was performed by the participant using shared autonomy control. This is done so that the participants could experience the shared autonomy control behavior before the actual tasks start.

The tasks of performing the different ADLs were given to the participants in different order to reduce the learning effect. Each task was performed 5 times where the involved objects were placed in different arbitrary positions. After finishing each task, the experiment participants filled a questionnaire rating the task performance and were able to comment on the task. The questionnaire filled by the users after each task is shown in Appendix A.2.1. After finishing the three tasks, participants were asked to fill a standard System Usability Scale (SUS) questionnaire rating the usability of the system, shown in Appendix A.2.2.

The different activities performed in this study were implemented as discussed in Section 4.5. The rotational axes of the SpaceMouse were disabled during the experiments to steer clear from any unwanted rotations that accompany the translations during operation.

For the pouring activity, the environment contained a bottle and a bowl placed on a desk. The users were asked to pick the bottle, pour its content into the bowl and place the bottle back on the desk. The relative positions of the bottle and the bowl were changed to examine placing the bottle close to and far away from the bowl as well as pouring from different directions by placing the bottle between the user and the bowl or placing the bowl between the user and the bottle.

For the contact guided motion, users were asked to wipe a window with a wiper. The user had to pick the wiper from a table top, make two swipes on the surface of the window, and then place the wiper again on the table after completion.

The locations of the window, the wiper, and the desk used in the window cleaning task were changed after each run, the window and the wiper were reoriented to have arbitrary angles each time. Allowing the wiper to be grasped from a different angle in each run, and would be re-oriented differently to align with the glass of the window each time the user approached the window to wipe it.

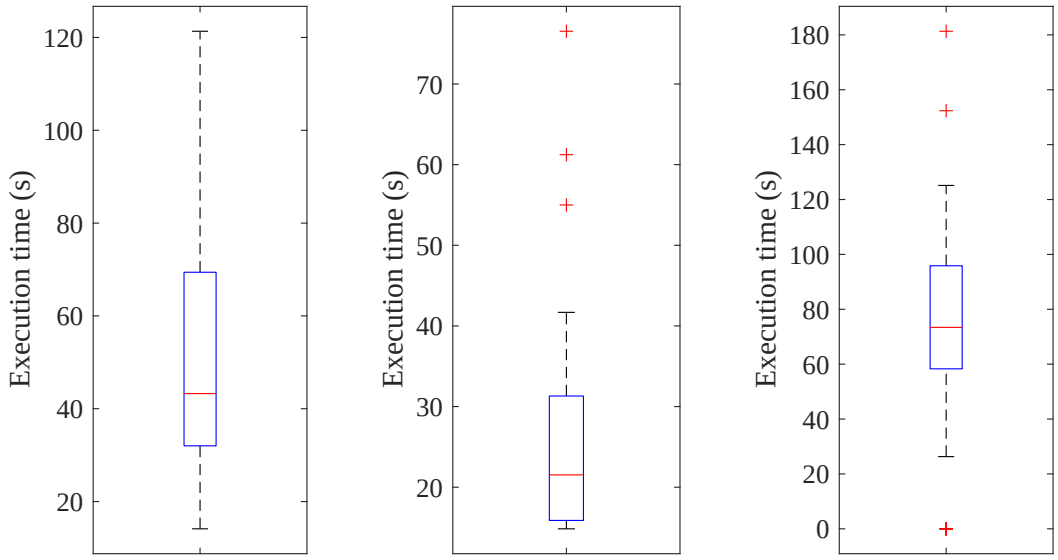
For the drawer opening activity, the users grasp the handle of a night-stand drawer, open the drawer, and release the handle. The location and orientation of the nightstand were also changed for each run. Different scenes showed the nightstand facing the user, facing away from the user, and facing sideways.

An experienced user performed the above ADLs by controlling the simulation of the robot arm using a SpaceMouse. The next section compares between the execution of the ADLs using both direct control and shared autonomy.

5.4 Results

This section describes the results of the experiments and compares between using the CARE framework for shared autonomy and using direct control. It shows examples of the different paths taken by the manipulator in response to the given user commands and the world state while performing different activities.

The average execution times recorded for the three tasks are shown in Table 1. Figure 34 represents the execution times for different tasks in box-plot diagrams showing the maximum and minimum as well as the median values for the time of each task.



(a) Pouring (b) Drawer opening (c) Window cleaning

Figure 34: The distribution of execution time for the different tasks. The plots show the median execution times (marked by the red lines) for each task. The boxes show the interquartile range, the middle 50% of execution times of the participants fall within this range. Lower and upper whiskers show lower and upper quartiles of execution times respectively. The outliers are denoted by the red corsses.

Activity	Average Time (sec)
Pouring	51.5
Drawer Opening	26.5
Wiping	86.6

Table 1: The average execution time (in seconds) of different activities using shared autonomy control.

For the pouring task, participants reported that the movement of the arm was intuitive with an average rating of 3.7/5. They also indicated that the system provided enough assistance to successfully complete the task.

During the task of opening the drawer, the participants gave an average rating of 4.2/5 for the intuitiveness of the arm motion. They indicated that the motion of the arm was natural and predictable. One of the participants reported feeling like they were fighting the system, this could be because they encountered a joint limit and the control algorithm was trying to adjust the configuration while the user tried to move the manipulator to complete the task.

During the window cleaning task, the results of the questionnaire indicate the users felt the system assisted them to finish the task, but more training time could have helped them to achieve better performance.

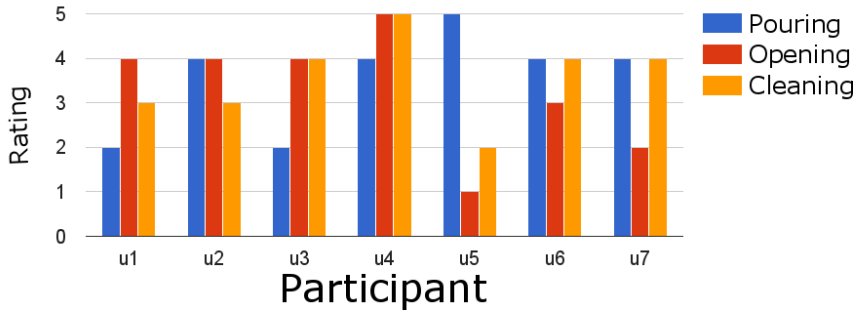
The SUS questionnaire filled by the users after each task shows that the CARE framework scored 63.6 on the SUS, which is slightly below average. This could be due to more than one factor.

The lack of stereo vision complicates the activities on the simulation in both the direct and shared autonomy control approaches, specially during the pick and place actions of the different objects. This is observed in direct control while trying to move the end effector to a grasping position, and then during the performance of the rest of the activities due to unintentional motion in unwanted directions, e.g. applying downward pressure on the SpaceMouse while pushing it forward will move the end effector down and forward simultaneously and thus, the user has to readjust the height of the end effector and for that they have to change the perspective in the simulation, which wasn't allowed during the experiments. However, Using the shared autonomy approach this is only observed until the end effector is in a position that triggers the constraints of an activity. After that, the user's commands are enhanced to facilitate the performance of the activity.

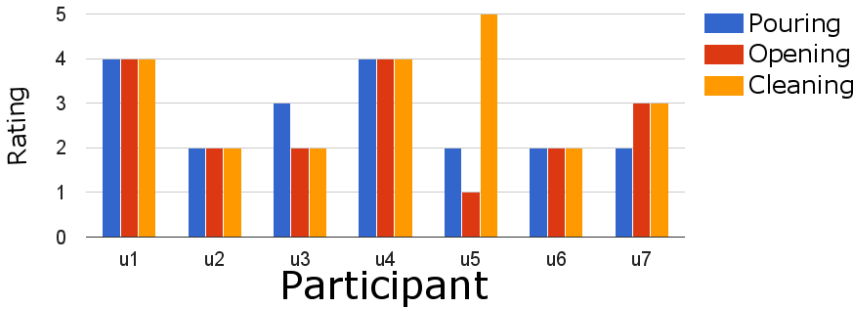
The participants had no prior experience with robot teleoperation, some of them ran into singularity configurations and joint limits while moving the arm. After recovering from these configurations the motion of the joints of the arm (joint space motion) might become unnatural because some joints are usually very close to their limits. Safety features exist that would avoid such situations by limiting the workspace of the arm, but they were disabled during the experiments.

This is indicated by the users' responses to questions 9 and 10 of the task evaluation questionnaire, Asking whether the system assisted the user to perform the task and whether they needed more training time respectively. The results for the three tasks is shown in Figure 35. Users who gave low responses for question 9 (Figure 35a) gave higher responses to question 10 (Figure 35b).

Picking a bottle using direct control, seen in Figure 36b can be a complex task requiring zig-zag motion to adjust the hand position for a correct and collision free grasp as well as lots of perspective changes (which weren't allowed in the user study) in order to see the correct location of the hand w.r.t the bottle and be able to move the hand accordingly. Such undulating motions aren't required in the shared autonomy case, because the framework handles the positioning (and orientation which is not visible in the case of the bottle) as soon as the hand moves into a position triggering one of the constraints for picking the bottle. This is shown in Figure 36a. The

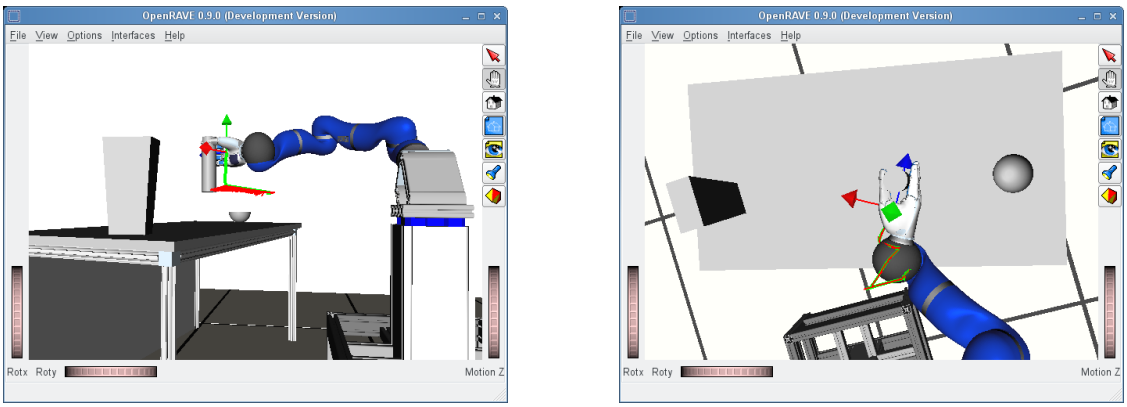


(a)



(b)

Figure 35: The path traveled by the end effector while performing different activities using shared autonomy and direct control.



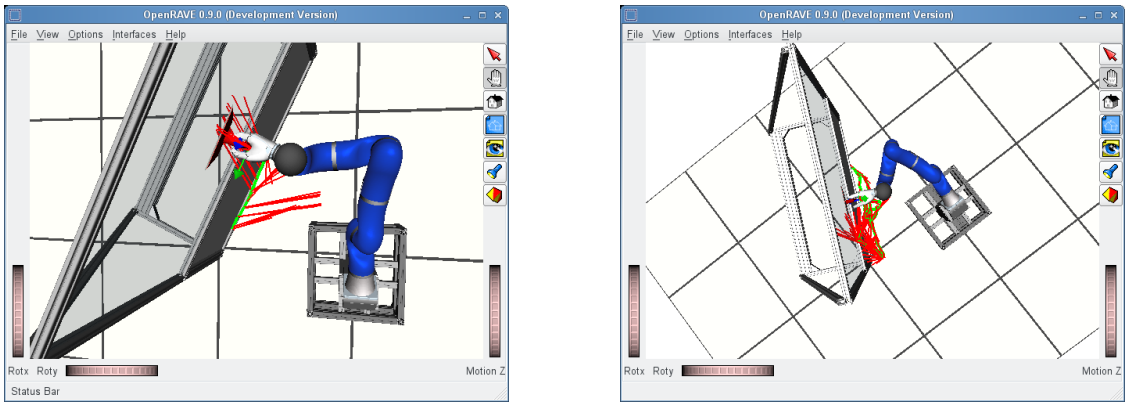
(a) (b)

Figure 36: The path traveled by the end effector to pick a bottle from a table top using shared autonomy and direct control. The actual paths taken by the manipulator in both cases are marked in green while the users commands at different points are marked in red. Figure 36a shows the path in case of the shared autonomy control. Figure 36a shows the path in case of direct user control, where the user follows a zig-zag motion in the direction of the bottle trying to steer the hand such that the fingers envelope the bottle at the end of the path. On the other hand, the green path in Figure 36a is made of two straight line segments where the user has to simply move the hand in the direction of the bottle and the CARE framework performs the necessary calculations to lead the hand to a grasping location (the horizontal segment) and then autonomously raises the bottle upwards in preparation for a next action (the vertical segment).

hand motion (marked in green) is formed of two straight line segments, the horizontal segment leads the hand to a grasp position even if the commands from the user, marked in red, aren't perfectly aligned with that path. The vertical segment was followed to raise the bottle above the table in preparation for the next action, this is added so that the user can move it freely after picking without worrying about collisions.

Figure 37 shows an example of teleoperating the robot arm to wipe a window using the CARE framework and direct control as seen in Figures 37a and 37b respectively. In the shared autonomy case, the motion is projected on a surface to keep the toolframe of the wiper in contact with the glass (Region of Interest) of the window, while the direct control case is only limited by collision avoidance and joint limits making it difficult to follow the surface (or apply pressure to it using the wiper). This is shown by the examining the green path in each of the figures as compared to the user commands shown in red.

Figure 38 shows the end effector path taken when it's guided by shared autonomy (Figures 38a and 38c) and when directly applying user's commands (Figures 38b and 38d). The paths taken by the end effector using direct control are following the user's commands throughout the duration of the activity. In contrast, shared autonomy enhances the user's commands to guide



(a) (b)

Figure 37: The path traveled by the end effector to wipe a window using shared autonomy and direct control. The actual paths taken by the manipulator in both cases are marked in green while the user commands at different points are marked in red. Figure 37a demonstrates the case of using shared autonomy to assist the user in following the surface of the window. This is observed by examining the user's commands in Figure 37a as compared to those in Figure 37b, in case of using the CARE framework, the commands are always projected on the surface such that the toolframe of the wiper is always in contact with the glass of the window while in case of direct control they follow the user in all cases except when they are limited to avoid collisions or joint limits.

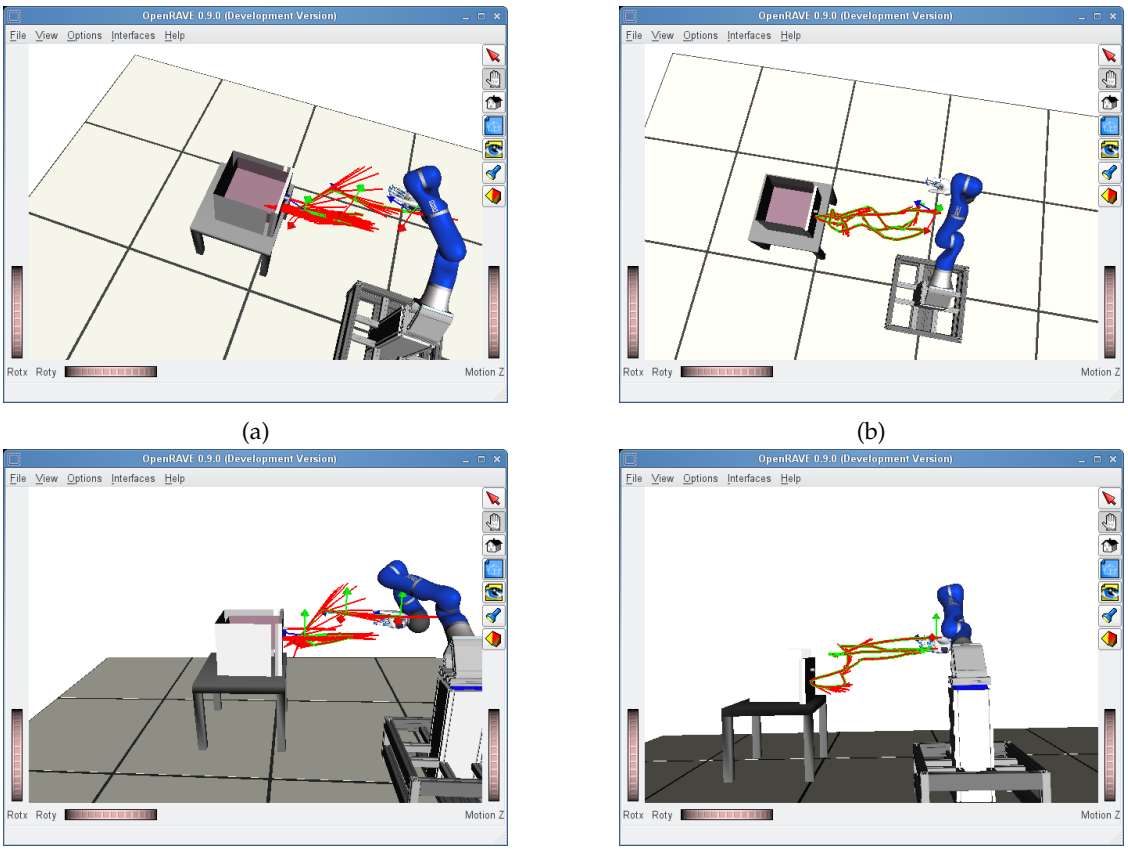


Figure 38: The path traveled by the end effector to pick the drawer-knob, open it, close it, and release it using shared autonomy and direct control. The actual paths taken by the manipulator in both cases are marked in green while the users commands at different points are marked in red. Figures 38a and 38c show the path in case of direct user control. Figures 38a and 38d show the path in case of the shared autonomy control. Unlike the direct control case, the difference between the red and green paths is more obvious in the shared autonomy case. The green paths contain straight line segments in the shared autonomy case, which are difficult to produce using direct control.

the end effector to a grasping point by projecting the commands on the drawer opening path till the drawer is open and then the drawer handle is released.

Although the perspective changes mentioned above are only crucial for running this software in a simulation, this behavior is still useful when performing ADL that involve partly occluded objects (hidden from the user's point of view).

6 Conclusion and Outlook

6.1 Summary

Relatively simple activities become impossible for people with tetraplegia without external help, they depend on the assistance of a caregiver for their daily needs. With the use of a Brain Computer Interface (BCI), those whose tetraplegia is caused by Spinal Muscular Atrophy (SMA) -and other degenerative illnesses- can control a robot arm to perform Activities of Daily Living (ADLs). However, the performance of these activities using direct control is uncomfortable and difficult due to difficulty in mapping a low dimensional input signal to a high degree of freedom robot arm in Cartesian space. Precise commands are needed to complete complex tasks, or those requiring high accuracy.

This work aims to assist people tetraplegia in controlling a robot arm using non-traditional input devices, such as a BCI, since they are unable to use the traditional ones, such as mice and keyboards. The work presented in this thesis focuses on giving them as much control as possible over the robot arm while at the same time minimally enhancing the input commands to perform different activities accurately and with ease.

This thesis presents the Constraint-based Activity REpresentation (CARE) framework, an object-centric constraint-based approach to describe manipulation actions. The approach assists the operator of a robot arm in a shared autonomy control paradigm -described in Section 3.1.3- to perform ADLs using a low dimensional input device.

In this framework, actions are divided into action phases, each of which is represented as a set of constraints. Constraints are categorized into the following three classes:

- Phase shift constraints.
- Geometric constraints.
- End effector constraints.

Phase shift constraints are used to determine which phase of the activity is currently active. The constraints of the currently active phase control the behavior of the robot arm. The geometric constraints help guide the motion of the end effector to successfully complete the activity. End effector constraints control the end effector configuration when needed, such as in the case of a pick or place activity. Using combinations of the different constraint classes, activities can be described in a way that makes teleoperation an intuitive and easy task.

To control a robot arm using this approach, a user only needs to take care of the general direction of the arm motion. The CARE framework ensures that actions are performed successfully by assisting the user in following their constraints. Actions are performed using the user's commands after enhancing them to follow certain paths or certain orientations of the robot's hand.

The framework provides an intention inference module used to infer the user's current activity. Based on this activity, the appropriate constraints are used according to which of the phases is active.

For the scope of this thesis, the work was done in the simulation where it is assumed that the robot knows the locations and types of the objects in the environment. The system uses the object handling system (world representation and object database) introduced by Leidner et al. [23] and has full knowledge of the environment. The inverse kinematics solver used for reachability checking is generated by OpenRAVE [8] and the arm is controlled using Cartesian impedance control [30, 42]. The work done used a dynamic simulation of the robot arm that offers realistic arm motion similar to the motion of the real arm.

Three ADLs are used to demonstrate and test how this approach can be applied to complex manipulation tasks, such as pouring a liquid from a container to another, cleaning a window with a wiper, and opening a drawer. The tests were performed using a SpaceMouse as a testbed simulating a low DoF input device. The CARE framework can be used in combination with other low DoF input devices such as a BCI and with other robotic manipulators since the system is independent of the type of robotic arm or manipulator.

Eventually, the developed method would enable people with tetraplegia using a robotic arm mounted on their wheel chair or bed and a low dimensional input device, such as a BCI. This set up shall be used to perform ADLs of varying complexity without external help. Such ADLs can span a wide range of activities ranging from scratching themselves to more complex ones such as pouring water and drinking without human assistance.

6.2 Future Work

For the experiments conducted in this work, a simulation of the robot arm was teleoperated by users in a virtual environment using a SpaceMouse. In addition, non of the participants has tetraplegia. The experiments need to be conducted on the real robot where participants with tetraplegia teleoperate the robot arm using a BCI to manipulate real objects. This might give a better insight to user requirements that will help in guiding future research in this field. This rest of this section suggests some improvements to the framework that would improve the proposed system.

6.2.1 Action Definitions

Some activities are more difficult to define than others as they require different levels of accuracy and detail in their description. This can be shown by considering the pouring activity. The pouring activity is divided into the three phases seen in Figure 12. While the users didn't report any abnormality in the execution of the pouring action, it does not the same as the execution performed by a human. Human would perform the motions involved in more than one phase simultaneously, like starting to rotate the bottle while still translating it in the direction of the

cup. Moreover, in the human execution, the transition between rotating around the frame and rotating around the toolframe is gradual as the rotation axis moves while the activity proceeds. This can be improved by using some techniques that learn manipulation tasks from human demonstration such as [6, 33].

The modularity of the CARE framework allows not only replacing the modules of the framework with others, but also replacing the callback functions for different constraints by others that may be more sophisticated or that lead to a more intuitive and transparent task execution. Replacing the callbacks involved in grasping, for example, with more intricate grasp planning techniques such as the approach introduced by Vogel et al. [43] would lead to a more accurate and intuitive pick action. [43] rates the grasps in the graspsets of different objects according to several factors in a preprocessing step and uses virtual fixtures at run time to guide the manipulator to the grasp of the highest rating.

6.2.2 Intention Inference

Some actions are hard to differentiate during their early stages. Symbolic reasoning used by the intention inference module, described in Section 4.4, is not enough to differentiate between them even after combination with the geometric information and adding reachability preconditions. For example, when an operator is moving a sponge in the direction of a table, it is difficult to predict whether the user is planning to clean/wipe the table or simply place the object on top of the table. Some heuristics are needed to differentiate between such activities. This could be solved by keeping a weighted history of the user's commands or commanded frames and using this to determine which object the user is actually moving towards and whether the user is moving towards the table or parallel to it.

One of the future milestones for the system presented here is the integration of a multi-step-plan recognition algorithm that uses both probability and symbolic information of the world, like the one introduced by Ramırez and Geffner [35]. To fully exploit this integration, more geometric information should be introduced into the action models to use them in the transition between actions in the plan. For instance, if the operator's long term plan is to prepare a bowl of cereal, this information can be used while placing the bowl by placing it in a location where pouring would be easier, i.e. a pouring frame would be reachable.

A Study Questionnaire

This study is to evaluate a new approach to shared autonomy teleoperation of a robotic arm. The approach is developed at the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt; DLR) and will be used to empower the people with Spinal Muscular Atrophy(SMA) -and other illnesses that prevent the movement and control of the arm- with the ability to perform Activities of Daily Living (ADLs) by controlling a robotic arm using a Brain Computer Interface (BCI).

A.1 Pre-Study Questions

1. Subject ID:
   First two Letters of mother's first name, first two Letters of the father's first name, and year of birth.

2. Age: _____

3. Gender:

Male	Female
○	○

4. I have used a SpaceMouse/ joystick before.

Yes	No
○	○

   - Shortly describe the task you used a SpaceMouse/Joystick for:

     _____

     _____

5. I have teleoperated/controlled a robot arm before.

Yes	No
○	○

   - What kind of input did you use for this?

     _____

     _____

   - Give a short description of the tasks you did with a robot arm?

     _____

     _____

A.2 System Evaluation Questions

A.2.1 Task Evaluation

- Please rate how much you agree/disagree with each of the following statements:

  1. I felt that the task was physically demanding.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  2. I felt that the task was mentally demanding.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  3. Teleoperating the arm was intuitive.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  4. Performing the task was difficult or frustrating.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  5. I needed some auditory/haptic feedback.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  6. I felt that the arm was unresponsive or that the motion should be faster than it was.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  7. The motion of the robot arm was natural.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  8. The actions were not predictable.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  9. The system assisted me to sufficiently finish the task; I didn't feel that I needed more help.

Strongly Agree	○	○	○	○	○	Strongly Disagree

  10. I needed more training time to finish the task better.

Strongly Agree	○	○	○	○	○	Strongly Disagree

- General Notes:
  Do you have any remarks that were not covered by this questionnaire?

A.2.2 System Usability Scale

- Please rate how much you agree/disagree with each of the following statements:

1. I think that I would like to use this system frequently.

Strongly Agree	○	○	○	○	○	Strongly Disagree

2. I found the system unnecessarily complex.

Strongly Agree	○	○	○	○	○	Strongly Disagree

3. I thought the system was easy to use.

Strongly Agree	○	○	○	○	○	Strongly Disagree

4. I think that I would need the support of a technical person to be able to use this system.

Strongly Agree	○	○	○	○	○	Strongly Disagree

5. I found the various functions in this system were well integrated.

Strongly Agree	○	○	○	○	○	Strongly Disagree

6. I thought there was too much inconsistency in this system.

Strongly Agree	○	○	○	○	○	Strongly Disagree

7. I would imagine that most people would learn to use this system very quickly.

Strongly Agree	○	○	○	○	○	Strongly Disagree

8. I found the system very cumbersome to use.

Strongly Agree	○	○	○	○	○	Strongly Disagree

9. I felt very confident using the system.

Strongly Agree	○	○	○	○	○	Strongly Disagree

10. I needed to learn a lot of things before I could get going with this system.

Strongly Agree	○	○	○	○	○	Strongly Disagree

References

[1] Bartels, G., Kresse, I., and Beetz, M. (2013). Constraint-based movement representation grounded in geometric features. In *13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 547–554.

[2] Birkenkampf, P., Leidner, D., and Borst, C. (2014). A knowledge-driven shared autonomy human-robot interface for tablet computers. In *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 152–159.

[3] Bischoff, R., Kurth, J., Schreiber, G., Koeppe, R., Stemmer, A., Albu-Schäffer, A., Eiberger, O., Beyer, A., Grunwald, G., and Hirzinger, G. (2010). Aus der forschung zum industrieprodukt: Die entwicklung des kuka leichtbauroboters. *at-Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik*, 58(12):670–680.

[4] Critchlow, A. J. (1985). *Introduction to robotics*. Macmillan New York.

[5] Dalibard, S., Nakhaei, A., Lamiraux, F., and Laumond, J.-P. (2010). Manipulation of documented objects by a walking humanoid robot. In *10th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 518–523.

[6] Dang, H. and Allen, P. K. (2010). Robot learning of everyday object manipulations via human demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1284–1289.

[7] Denavit, J. and Hartenberg, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 22:215–221.

[8] Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute.

[9] Fahimi, F. (2009). *Autonomous Robots: Modeling, Path Planning, and Control*, chapter Redundant Manipulators, pages 1–36. Springer US, Boston, MA.

[10] Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D. E., et al. (1998). Pddl-the planning domain definition language.

[11] Goodrich, M. A., Crandall, J. W., and Barakova, E. (2013). Teleoperation and beyond for assistive humanoid robots. *Reviews of Human Factors and Ergonomics*, 9(1):175–226.

[12] Green, A. M. and Kalaska, J. F. (2011). Learning to move machines with the mind. *Trends in neurosciences*, 34(2):61–75.

[13] Hart, S., Dinh, P., and Hambuchen, K. (2014). Affordance templates for shared robot control. In *Artificial Intelligence and Human-Robot Interaction, AAAI Fall Symposium Series, Arlington, VA. USA*.

[14] Hirzinger, G. (1999). Intuitive robot motion control. *Journal of the Robotics Society of Japan*, 17(2):175–179.

[15] Hochberg, L. R., Bacher, D., Jarosiewicz, B., Masse, N. Y., Simeral, J. D., Vogel, J., Haddadin, S., Liu, J., Cash, S. S., van der Smagt, P., et al. (2012). Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, 485(7398):372–375.

[16] Israelsen, J., Beall, M., Bareiss, D., Stuart, D., Keeney, E., and van den Berg, J. (2014). Automatic collision avoidance for manually tele-operated unmanned aerial vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6638–6643.

[17] James, J., Weng, Y., Hart, S., Beeson, P., and Burridge, R. (2015). Prophetic goal-space planning for human-in-the-loop mobile manipulation.

[18] Jetchev, N. and Toussaint, M. (2009). Trajectory prediction: learning to map situations to robot trajectories. In *Proceedings of the 26th annual international conference on machine learning*, pages 449–456. ACM.

[19] Katz, D., Venkatraman, A., Kazemi, M., Bagnell, J. A., and Stentz, A. (2014). Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. *Autonomous Robots*, 37(4):369–382.

[20] Kucuk, S. and Bingul, Z. (2006). *Robot kinematics: forward and inverse kinematics*. INTECH Open Access Publisher.

[21] Lana, E. P., Adorno, B. V., and Maia, C. A. (2015). A new algebraic approach for the description of robotic manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3083–3088.

[22] Le, Q. V., Kamm, D., Kara, A. F., and Ng, A. Y. (2010). Learning to grasp objects with multiple contact points. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5062–5069.

[23] Leidner, D., Borst, C., and Hirzinger, G. (2012). Things are made for what they are: Solving manipulation tasks by using functional object classes. In *12th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 429–435.

[24] Liu, H., Meusel, P., Seitz, N., Willberg, B., Hirzinger, G., Jin, M., Liu, Y., Wei, R., and Xie, Z. (2007). The modular multisensory dlr-hit-hand. *Mechanism and Machine Theory*, 42(5):612 – 625.

[25] Maheu, V., Archambault, P. S., Frappier, J., and Routhier, F. (2011). Evaluation of the jaco robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities. In *IEEE International Conference on Rehabilitation Robotics*, pages 1–5.

[26] Marayong, P., Li, M., Okamura, A. M., and Hager, G. D. (2003). Spatial motion constraints: Theory and demonstrations for robot guidance using virtual fixtures. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1954–1959.

[27] Martinez-Perez, F. E., Fraga, J., and Tentori, M. (2010). Artifacts' roaming beats recognition for estimating care activities in a nursing home. In *4th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth).*, pages 1–8. IEEE.

[28] Martínez-Pérez, F. E., González-Fraga, J. Á., Cuevas-Tello, J. C., and Rodríguez, M. D. (2012). Activity inference for ambient intelligence through handling artifacts in a healthcare environment. *Sensors*, 12(1):1072–1099.

[29] Muelling, K., Venkatraman, A., Valois, J.-S., Downey, J., Weiss, J., Javdani, S., Hebert, M., Schwartz, A. B., Collinger, J. L., and Bagnell, J. A. (2015). Autonomy infused teleoperation with application to bci manipulation. *arXiv preprint arXiv:1503.05451*.

[30] Ott, C. (2008). *Cartesian impedance control of redundant and flexible-joint robots*. Springer.

[31] Paul, R. P. (1981). *Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators*. Richard Paul.

[32] Petit, D., Gergondet, P., Cherubini, A., and Kheddar, A. (2015). An integrated framework for humanoid embodiment with a bci. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2882–2887.

[33] Phillips, M., Hwang, V., Chitta, S., and Likhachev, M. (2016). Learning to plan for constrained manipulation from demonstrations. *Autonomous Robots*, 40(1):109–124.

[34] Ramırez, M. and Geffner, H. (2009). Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, pages 1778–1783.

[35] Ramırez, M. and Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1121–1126. Citeseer.

[36] Saxena, A., Driemeyer, J., and Ng, A. Y. (2008). Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173.

[37] Schroer, S., Killmann, I., Frank, B., Volker, M., Fiederer, L., Ball, T., and Burgard, W. (2015). An autonomous robotic assistant for drinking. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6482–6487.

[38] Sheridan, T. B. (1992). *Telerobotics, automation, and human supervisory control*. MIT press.

[39] Sung, J., Ponce, C., Selman, B., and Saxena, A. (2012). Unstructured human activity detection from rgbd images. In *10th IEEE-RAS International Conference on Robotics and Automation (ICRA)*, pages 842–849.

[40] Tian, X., Zhuo, H. H., and Kambhampati, S. (2015). Discovering underlying plans based on distributed representations of actions. *arXiv preprint arXiv:1511.05662*.

[41] Vogel, J., Bayer, J., and Van Der Smagt, P. (2013). Continuous robot control using surface electromyography of atrophic muscles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 845–850.

[42] Vogel, J., Haddadin, S., Jarosiewicz, B., Simeral, J. D., Bacher, D., Hochberg, L. R., Donoghue, J. P., and van der Smagt, P. (2015). An assistive decision-and-control architecture for force-sensitive hand–arm systems driven by human–machine interfaces. *The International Journal of Robotics Research*, 34(6):763–780.

[43] Vogel, J., Hertkorn, K., Menon, R. U., and Roa, M. A. (2016). Flexible, semi-autonomous grasping in assistive robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[44] Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., and Vaughan, T. M. (2002). Brain–computer interfaces for communication and control. *Clinical neurophysiology*, 113(6):767–791.

[45] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438.